PARALLEL HYBRID OPTIMIZATION METHODS FOR PERMUTATION BASED PROBLEMS

Malika Mehdi

PhD-FSTC-2011-17
Faculté des Sciences, de la Technologie et de la Communication

Faculté des Sciences et Technologies de Lille

THÈSE

Soutenue le 20/10/2011 à Luxembourg
En vue de l'obtention du grade académique de

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

ET

DOCTEUR DE L'UNIVERSITÉ LILLE1

EN INFORMATIQUE

par

Malika Mehdi
née le 10 septembre 1981 à Bejaia

PARALLEL HYBRID OPTIMIZATION METHODS FOR PERMUTATION BASED PROBLEMS

Jury de thèse

Prof . Dr. Pascal Bouvry, directeur de thèse
Professeur, Université du Luxembourg

Prof. Dr. El-Ghazali Talbi, directeur de thèse
Professeur, Université Lille1

Prof. Dr. Raymond Bisdorff, président
Professeur, Université du Luxembourg

Prof. Dr. Nouredine Melab, président suppléant
Professeur, Université Lille1

Prof. Dr. Enrique Alba, membre
Professeur, Université de Malaga

Dr. Didier El-Baz, membre
HDR, LAAS-CNRS, Toulouse

Abstract

Solving efficiently large benchmarks of NP-hard permutation-based problems requires the development of hybrid methods combining different classes of optimization methods. Indeed, it is now acknowledged that such methods perform better than traditional optimization methods when used separately. The key challenge is how to find connections between the divergent search strategies used in each class of methods in order to build efficient hybridization strategies. Genetic algorithms (GAs) are very popular population-based metaheuristics based on stochastic evolutionary operators. The hybridization of GAs with tree-based exact methods such as Branch-and-Bound is a promising research trend. B&B algorithms are based on an implicit enumeration of the solution space represented as a tree. Our hybridization approach consists in providing a common solution and search space coding and associated search operators enabling an efficient cooperation between the two methods.

The tree-based representation of the solution space is traditionally used in B&B algorithms to enumerate the solutions of the problem at hand. In this thesis, this special representation is adapted to metaheuristics. The encoding of permutations as natural numbers, which refer to their lexicographic enumeration in the tree, is proposed as a new way to represent the solution space of permutation problems in metaheuristics. This encoding approach is based on the mathematical properties of permutations (Lehmer codes, inversion tables, etc.). Mapping functions between the two representations (permutations and numbers) and special search operators adapted to the encoding are defined for general permutation problems, with respect to the theory of representation. This common representation allows the design of efficient cooperation strategies between GAs and B&B algorithms. In this thesis, two hybridization schemes combining GAs with B&B based on this common representation are proposed. The two hybridization approaches HGABB/HAGABB (Hybrid Adaptive GA-B&B) and COBBIGA (cooperative B&B interval-based GA), have been validated on standard benchmarks of one of the hardest permutation-based problems, the three dimensional quadratic assignment problem (Q3AP). In order to solve large benchmarks of permutation-based problems, a parallelization for computational grids is also proposed for the two hybrid schemes. This parallelization is based on space decomposition techniques (the decomposition by intervals) used in parallel B&B algorithms. From the implementation point of view, in order to facilitate further design and implementation of hybrid methods combining metaheuristics with tree-based exact methods, a hybridization C++ framework integrated to the framework for metaheuristics ParadisEO is developed. The new framework is used to conduct extensive experiments over the computational grid Grid'5000.

Resumé

La résolution efficace de problèmes d'optimisation à permutation de grande taille nécessite le développement de méthodes hybrides complexes combinant différentes classes d'algorithmes d'optimisation. L'hybridation des metaheuristiques avec les méthodes exactes arborescentes, tel que l'algorithme du branch-and-bound (B&B), engendre une nouvelle classe d'algorithmes plus efficace que ces deux classes de méthodes utilisées séparément. Le défi principal dans le développement de telles méthodes consiste à trouver des liens ou connections entre les stratégies de recherches divergentes utilisés dans les deux classes de méthodes. Les Algorithmes Génétiques (AGs) sont des metaheuristiques, à base de population, très populaires basés sur des opérateurs stochastiques inspirés de la théorie de l'évolution. Contrairement aux AGs et aux métaheuristiques généralement, les algorithmes de B&B sont basés sur l'énumération implicite de l'espace de recherche représenté par le moyen d'un arbre, dit arbre de recherche. Notre approche d'hybridation consiste à définir un codage commun des solutions et de l'espace de recherche ainsi que des opérateurs de recherche adéquats afin de permettre un couplage efficace de bas niveau entre les deux classes de méthodes AGs et B&B.

La représentation de l'espace de recherche par le moyen d'arbres est traditionnellement utilisée dans les algorithmes de B&B. Dans cette thèse, cette représentation a été adaptée aux metaheuristiques. L'encodage des permutations au moyen de nombres naturels faisant référence à l'ordre d'énumération lexicographique des permutations dans l'arbre du B&B, est proposé comme une nouvelle manière de représenter l'espace de recherche des problèmes à permutations dans les metaheuristiques. Cette méthode de codage est basée sur les propriétés mathématiques des permutations, à savoir les codes de Lehmer et les tables d'inversions ainsi que les système d'énumération factoriels. Des fonctions de transformation permettant le passage entre les deux représentations (permutations et nombres) ainsi que des opérateurs de recherche adaptés au codage, sont définis pour les problèmes à permutations généralisés. Cette représentation, désormais commune aux metaheuristiques et aux algorithmes de B&B, nous a permis de concevoir des stratégies d'hybridation et de collaboration efficaces entre les AGs et le B&B. En effet, deux approches d'hybridation entre les AGs et les algorithmes de B&B (HGABB et COBBIGA) basés sur cette représentation commune ont été proposées dans cette thèse. Pour validation, une implémentation a été réalisée pour le problème d'affectation quadratique à trois dimension (Q3AP). Afin de résoudre de larges instances de ce problème, nous avons aussi proposé une parallelisation pour les deux algorithme hybrides, basée sur des techniques de décomposition d'espace (décomposition par intervalle) utilisées auparavant pour la parallelisation des algorithmes de B&B. Du point de vue implémentation, afin de faciliter de futurs

conceptions et implémentations de méthodes hybrides combinant metaheuristiques et méthodes exacte arborescentes, nous avons développé une plateforme d'hybridation intégrée au logiciel pour metaheuristiques, ParadisEO. La nouvelle plateforme a été utilisée pour réaliser des expérimentations intensives sur la grille de calcul Grid'5000.

To my parents
To my brothers
To my husband Ahmed

Acknowledgements

Durant ces quatre années de thèse, j'ai bénéficié de l'encadrement, support et encouragement de nombreuses personnes en Algérie, à Lille et à Luxembourg que j'aimerais remercier dans ces lignes. Je commence par exprimer ma grande gratitude à mes trois co-directeurs de thèse Pascal Bouvry, El-Ghazali Talbi et Nouredine Melab, c'est grâce à votre expérience et vos encouragements que j'ai pu mener mes travaux de recherche dans le bon sens malgré les moments de doute. J'ai appris énormément de choses avec chacun de vous durant ces quatre années tant sur le plan professionnel que personnel. C'était un plaisir de travailler avec vous.

Je tiens aussi à remercier Raymond Bisdorff en tant que président de jury et surtout en tant que membre du comité d'encadrement de ma thèse avec qui j'ai eu l'occasion de travailler durant les réunions annuelles du comité de thèse. J'aimerais aussi remercier Enrique Alba et Didier El-Baz pour avoir accepté de reviewer ce travail et d'y consacrer un peu de leur temps ainsi que pour leurs nombreuses remarques fructueuses sur la façon d'améliorer le manuscrit de thèse.

Durant cette thèse j'ai aussi eu l'occasion de travailler avec Bertrand Le Cun, François Galea et Van dat Cung (Université de Versailles) que j'aimerais remercier ici de m'avoir fourni le logiciel BOB++. Dans le même cadre, je remercie également Mohand Mezmaz et Jean-Claude Charr avec qui j'ai travaillé sur la parallélisation des approches hybrides pour le Q3AP. Ces collaborations m'ont permis d'avancer dans mes recherches.

Sur un autre plan, j'aimerais remercier tout les membres de l'équipe à Luxembourg et à Lille. Merci beaucoup Julien, Emilia, Alexadru, Gregoire, Cesar, Jakub, Johnathan, Patricia, Bernabé, See, Thomas, Benoit, Sebastien, Marçin, Guillaume, Frederic, Agata et tout les nouveaux membres. Merci à chacun d'entre vous pour votre support chacun à sa manière: Discussions et échanges sur les algorithmes révolutionnaires, les permutations et autres sujets (notamment avec Gregoire, Emilia, Alexandru, Johnathan, et Sebastien), tournées de Billard à la "caffette", éviter de me poser la question qui fâche trop souvent :), des mots d'encouragement durant la période de rédaction, etc. Enfin, je tiens à remercier tout particulièrement ma collègue de bureau Emilia, merci beaucoup pour ta gentillesse et ton aide même après mon départ de Luxembourg.

A Lille également, j'aimerais remercier tout les membres de l'équipe DOLPHIN; les permanents d'être toujours disponibles pour échanger quelques discussions,

et les doctorants pour l'ambiance fraternelle qui y règne. Je tiens à remercier tout particulièrement Yacine, Karima, les deux Mustapha, Mathieu, Marie et Thé Van. Je n'ai eu que de courts passages à Lille durant ces deux dernières années, mais c'était toujours un plaisir de vous revoir et de discuter de nos vies de thésards ! Je souhaite bonne continuation à ceux qui ont déjà soutenue et bonne chance pour Yacine, Mustapha et Mathieu.

Je n'oublie pas également de remercier le bureau secrétariat à Luxembourg, Isabelle et Fabienne, à Lille également un grand merci à Tram secrétaire de l'école doctorale pour sa disponibilité.

Sur un titre personnel, je remercie mes amis et collègues du SNT à Luxembourg avec qui j'ai partagé du bon temps à travers des sorties et diners organisés en toutes occasions. Dalia et Djamila ! Je ne saurais vous remercier pour votre amitié et votre gentillesse. J'ai passé d'agréables moments en votre compagnie et je suis très reconnaissante pour votre support durant ma thèse. Un grand merci également à Iram, Donia et Ayda, merci pour tout et je vous souhaite bonne chance pour la suite.

Mes derniers remerciements et non les moindres vont à ma famille en France et en Algérie en particulier à mes parents pour leur soutient inconditionnel durant tout mon cursus universitaire et à mes grands parents à Lille qui m'ont donné la chance de démarrer cette aventure. Enfin, Ahmed, tout ce travail de thèse repose en grande partie sur ta compréhension, ton support et tes encouragements et ceci dés le début et jusqu'au dernier moment, ce n'est plus du support mais de la vrai collaboration ! Merci.

Je termine par adresser mes remerciement à toute personne ayant aidé de loin ou de prêt à la réalisation de ce travail de thèse et à tout ceux qui ont participé à ma formation.
Cette thèse a été financée par le FNR Luxembourg pour lequel je suis également reconnaissante.

Malika.

Contents

Chapter 1

Introduction

Contents

1.1 Context

Permutations are used in many combinatorial optimization problems (COPs) to represent candidate solutions. Such problems are commonly known as permutation-based problems. Permutation-based problems can be found in various application areas such as scheduling and sequencing, design of VLSI circuits, campus design, routing, etc. They consist either of the optimization of the assignment cost between two sets of objects of the same cardinality, or the optimization of a schedule of jobs or any other planing application represented as a sequence of tasks, and so on. Such problems are generally NP-hard [42] and some of them are among the most challenging optimization problems in terms of resolution time. Their complexity is due partially to the exponential size of their search space, which makes large benchmarks very difficult to be solved.

This thesis takes place in the context of the new class of combinatorial search algorithms, hybrid search algorithms, which consists in combining different search algorithms and techniques in order to efficiently solve larger instances of optimization problems. Particularly, the hybridization of some tree-based exact search methods such as Branch-and-Bound with metaheuristics is a new promising trend in the optimization community. Indeed, the differences in the search strategies adopted in each class of methods offers a great opportunity to compensate the weaknesses of each class by the points of strength of the other class. The Branch-and-Bound algorithm is an enumerative deterministic algorithm which uses a tree to represent the set of all possible configurations to the problem to be solved. When applied to permutations, the search space is organized as a permutation tree and the algorithm implicitly enumerates all the feasible space in order to find the optimal solution(s) and prove their optimality.

Alternatively, metaheuristics are general purpose approximate search techniques which aim at finding near-optimal solutions to the problem at hand. Evolutionary algorithms (EAs) such as Genetic algorithms (GAs) are very popular population-based metaheuristics inspired from the theory of evolution. A population of initial configurations called individuals is evolved through generations by the use of some genetic operators by analogy to their counterparts in biological systems: selection, crossover, mutation, etc. These operators are partially based on random numbers. Thus, the search strategies used in tree-based exact algorithms and metaheuristics are very divergent but complementary. This makes the combination of these two classes of methods a very promising research area for solving large permutation-based problems.

On the other hand, such complex algorithms are CPU time intensive. The evolution of parallel architectures leads to the development of a broad range of parallelization strategies for both exact search algorithms and metaheuristics. The work carried out in this thesis can also be placed in the context of parallelization strategies applied to complex hybrid algorithms combining metaheuristics with tree-based exact methods.

1.2 Motivations

GAs [44] have been combined with different optimization methods (approximate or exact ones) in order to obtain a new class of methods, that explore more efficiently the search space of large and complex combinatorial optimization problems. The basic idea of such hybrids is to take advantage from the complementary behaviors of the methods they combine.

In this thesis, the focus is set on combining metaheuristics with the Branch-and-Bound algorithm (B&B) [65]. The latter uses a tree structure to progressively enumerate the search space and uses relaxation techniques to eliminate sub-trees that are not likely to lead to optimal solutions.

If we consider the hybridization of GAs with B&B, relevant ways to combine these two algorithms have been largely discussed in the literature and successfully applied to different classes of optimization problems ([87], [41], [21], [34]). Those methods could be classified in two categories: relay methods working in a sequential mode (the input of one method is the output of the previous method) or cooperative methods where the two algorithms are executed in a parallel or intertwined way and exchange information to guide their respective searches. Generally, the main information exchanged in a cooperative hybridization between GAs and B&B is new best found solutions and the initialization of the GA's population from the B&B's pool of nodes. However, the advantages took from the cooperation between GA and B&B could go beyond the traditional weakness/strength balance, best solutions exchanges, etc., if the characteristics of the two algorithms are exploited to find some "connections" between their respective searches.

The issue addressed in this thesis is how to build efficient low-level and high-level hybrid cooperative optimization methods that combine metaheuristics with B&B or with any tree-based exact search. We believe that the use of the same representation in both sides will help to offer new ways of efficient communications and will allow accurate information exchange in hybrid cooperative methods.

Recently, a special encoding of the B&B tree is used to parallelize the B&B algorithm in [76]. This encoding consists in assigning a unique number to each node in the tree. In the

leaves, this number represents the sequence number of each solution during the enumeration process. Each group of contiguous nodes forms an interval. This transforms the permutation tree into a simple one dimensional space represented as an interval that covers all the solution space. The generalization of this representation to metaheuristics allows the latters to exploit the advantages of the enumeration process used in B&B. For example, diversification and intensification tools may be designed with regard to the tree. On the other hand, hybridization between metaheuristics and B&B will be more efficient because the two algorithms are aware about each other search. Pertinent information can be exchanged to indicate which part of the tree is already explored and which part remains to be explored. A third advantage of the generalization of the encoded tree to metaheuristics is the parallelism. The interval concept is used initially to design efficient parallelization strategies for B&B over a computational grid [76]. This parallelization strategy can be easily generalized to metaheuristics and to hybrids between metaheuristics and B&B.

The generalization of this encoded tree to metaheuristics requires the use of some mathematical properties of permutations and some algorithms from the discrete mathematics community, such as Lehemer codes, inversion tables, factorial number systems, etc.

1.3 Thesis Contributions

The contributions of this thesis can be summarized in the following points.

Classification of the most known permutation-based problems

In this dissertation, a review of the most known permutation-based problems in different fields (assignment, scheduling, routing) and their classification according to the interpretation made of the permutation encoding, is proposed. This study is completed with a review of some state-of-the-art neighborhood and genetic operators used for each class of permutation problems. Such classification is needed for the design of hybrid metaheuristics for this class of problems. Indeed, different genetic and neighborhood operators for permutations are proposed in the literature but not all of them are adequate for all permutation-based problems. In this dissertation, the proposed classification is based on different works on the subject from the literature. In [56], Kargupta *et al.* presented a study of crossover operators for *deceptive ordering problems*. Ordering or sequencing problems are generally represented by permutations but not all genetic operators applied to permutations are efficient for sequencing problems. Bierwirth also pointed out in [10] that crossover operators used for assignment problems are not adequate for the Jobshop problem (JSP) and argued that efficient operators for this problem should be respectful of the constraints of the problem: the relative order between jobs in a schedule and the precedence constraints. Nevertheless, to our knowledge there is not a complete work dedicated to permutation problems in which the reader may learn how to distinguish between the different neighborhood and genetic operators proposed for different types of permutation-based problems. Such a study may help to efficiently reuse existing operators for new permutation-based problems or the design of problem-specific operators if required.

Design of parallel low-level and high level hybridization schemes combining GAs with B&B

In order to tackle large instances of challenging permutation-based problems, during this thesis the focus was set on hybrid optimization search methods which combine different types of algorithms. The particularity of permutation-based problems is their exponentially growing search spaces when the size of the instances to be solved increases. Thus, our objective consists in combining GAs with tree-based search methods such as B&B in order to take advantage from their divergent search strategies and efficiently explore large search spaces of permutation-based problems.

The hybridization strategies proposed in this thesis consist in using a common representation of the solution space in the two methods in order to allow a tight and efficient cooperation. The B&B algorithm uses a tree to enumerate the solution space of the problem at hand. Our idea consists in using the same representation in GAs. This common guidance of the solution space allows the two algorithms to be aware about each other search. Thus, it enables them to exchange fruitful information about: new best-found solutions, promising sub-spaces to search intensively (from GA to B&B) and unvisited sub-spaces to explore (from B&B to GA). Two hybridization schemes based on this idea are proposed. The Hybrid algorithm GA-B&B (HGABB) and its adaptive version Hybrid Adaptive GA-B&B is a low level hybridization strategy in which a B&B is embedded in a mutation operator in order to optimally search large subspaces. Some mechanisms from other metaheuristics are used to escape optimally solved subspaces when the GA reaches convergence. The second scheme is a high level asynchronous cooperative B&B interval-based GA (COBBIGA). COBBIGA is a tightly coupled cooperative method based on some space decomposition techniques used recently in B&B algorithms. Indeed, the concept of interval has been proposed in [76] and used to parallelize B&B algorithms for permutation problems. It consists in assigning a unique number to each node in the B&B tree and the use of intervals to represent contiguous nodes in the tree. The encoding technique is based on the structure of the permutation tree. In COBBIGA, the concept of interval is generalized to GAs in order to encode the information exchange already explained with the B&B algorithm.

Such complex hybrid methods are CPU time intensive. Thus, we also proposed a parallelization of the the hybrid schemes HGABB/HAGABB and COBBIGA. The proposed parallelization strategies are based on space decomposition techniques inspired from parallelization strategies of the B&B algorithm and other tree-based search algorithms.

Definition of adequate operators to search restricted subspaces in metaheuristics

In COBBIGA, the GA frequently receives information from the B&B about unvisited subspaces to be explored. On the other hand, some sub-spaces have already been optimally solved by B&B. Thus, these sub-spaces should be ignored by the GA's search. This information exchange is encoded using intervals. Each interval covers a set of nodes in the B&B tree. In order to restrict the search of GAs to a given interval, we need special type of search operators, crossovers, mutations and neighborhood operators that respect the interval. To solve

this issue, we defined a class of search operators called interval-based operators (I-operators) that allows GAs and metaheuristics generally to focus their search on a given subspace. Furthermore, the new operators are designed in such a way to respect the main properties of representations and neighborhoods: the locality and the connectivity respectively.

Mapping algorithms for generalized permutation-based problems

The encoding of permutations by natural numbers in metaheuristics is a recent idea used by few researchers [94], [82]. Each permutation on $[n]$ is represented by a unique number which represents its order among the $n!$ possible permutations enumerated following a lexicographic order. Such representation is used in [82] and [94] to simplify some search mechanisms in metaheuristics by simplifying the search space from a complex permutation solution space to a continuous one. Mapping functions between the space of permutations and the natural numbers in the interval $[0, n![$ already exist in the discrete mathematics community. These mapping functions are based on some mathematical properties of permutations and factorial number systems. In the context of exact optimization, the same idea is used in [76] to encode the nodes in a B&B tree. The structure of the permutation tree is used to this purpose and mapping functions are defined between the nodes in the tree and numbers of the interval $[0, n![$. In this thesis, we are interested on generalized permutation-based problems which use more than one permutation to represent potential solutions. Thus, new mapping operators from the space of multidimensional permutations to numbers are proposed. Those operators are based on the mathematical properties of permutations (Lehmer codes, inversions), factorial number systems and permutation trees. Such functions are needed in the metaheuristic community in order to generalize the new representation to all permutation-based problems. Furthermore, they could be useful in other domains where permutation are used, for other purposes.

ParadisEO-HEM: a hybridization framework for the development of hybrids combining metaheuristics with exact methods

In order to facilitate further design and implementation of hybridization strategies combining metaheuristics with exact methods, a hybridization framework integrated to the software for metaheuristics ParadisEO [17] is developed. ParadisEO-HEM (Hybrid Exact Metaheuristics) is a C++ framework which can be used to implement parallel hybrid search algorithms combining metaheuristics developed using ParadisEO and exact search algorithms developed in any other dedicated software. The parallel hybrid algorithms proposed in this thesis are developed using this framework.

A case study on the three dimensional quadratic assignment problem (Q3AP)

In order to validate the parallel hybrid methods COBBIGA and HGABB/HAGABB, an implementation is proposed for Q3AP. Extensive experiments were conducted over the computational grid Grid'5000. Large benchmarks of this problem are tackled and new near-optimal solutions are found for unsolved benchmarks. This part of the thesis was done in

the context of the French ANR project CHOC [1](challenges for Hybrid Combinatorial Optimization) in collaboration with PRISM [2] laboratory at Versailles University which provided us with a B&B solver for Q3AP [40].

Because of the huge exponential search space of this problem, the largest Q3AP benchmark solved to optimality in the literature was limited to 14, as reported by Hahn *et al* in a recent review [48]. More recently, Galea and Le Cun in [39] proposed a parallelization for shared memory machines of a B&B algorithm for Q3AP, based on the parallelization of the lower-bound function. This pushed up the size of the largest Q3AP benchmark solved to optimality to size 16. Nevertheless, typical real world applications for this problem are of size 16, 32 and 64. In this thesis, we tackled benchmarks until size 20 for which new near-optimal solutions are found. Those solutions could be used as upper-bounds in B&B algorithms for Q3AP. We also conducted some experiments using the parallel COBBIGA algorithm deployed on hundreds of processing cores in Grid'5000, to solve a Q3AP benchmark of size 15 (Nug15 derived from the QAP benchmark of the same name [14]). This experiment lasted 9 days, 11 hours and 40 minutes.

Finally, in collaboration with M. Mezmaz (Mons University, Belgium), we adapted the platform B&B@Grid [77] for grid enabled parallel B&B algorithms to the Q3AP. This platform was validated when solving hard benchmarks of another permutation problem, the flowshop scheduling problem. During this thesis, B&B@Grid for Q3AP was used to conduct experiments on thousands of processing cores belonging to Grid'5000. The objective of these experiments was to confirm the optimal solution found using COBBIGA for the benchmark Nug15.

1.4 Outline of the thesis

The rest of the thesis is organized in two parts. **Part I** is a general introduction to combinatorial optimization and permutation-based optimization problems, and is composed of two chapters. **Part II** contains the contributions of this thesis composed of three chapters. The manuscript is ended with some concluding remarks and future research directions.

Part I: Combinatorial optimization for permutation-based problems

Chapter 2 addresses a general view on exact and heuristic methods for NP-hard combinatorial optimization problems. Tree-based exact algorithms such as the Branch-and-Bound (B&B) are introduced as well as some metaheuristic algorithms. The chapter also addresses hybridization strategies and parallelization techniques for metaheuristics and tree-based search algorithms.

Chapter 3 is an introduction to permutation-based problems. A non exhaustive list of permutation-based problems is given. This review is completed by an analysis of the interpretation of the permutation encoding for each problem and the state-of-the-art search operators used for each category of permutation-based problems. Some mathematical properties of permutations and tools are also presented.

1. http://choc.prism.uvsq.fr
2. http://www.prism.uvsq.fr

Part II: Thesis contributions

This part of the dissertation contains three chapters which summarize the contributions of this thesis.

In **Chapter 4**, the new mapping algorithms between the space of permutations and the natural numbers for general permutation-based problems are explained. The concept of intervals used in the B&B is revisited in metaheuristics and some interval-based local search operators are defined with respect to the concepts of locality of a representation and connectivity of neighborhoods.

Chapter 5 describes the two hybridization schemes combining GAs with tree-based search algorithms (B&B), the algorithms HGABB/HAGABB and COBBIGA. The two schemes are based on an explicit decomposition of the search space using the concept of interval. To validate the two hybrid algorithms, an implementation is done together with their experimentation for Q3AP.

In **Chapter 6** the parallelization over a computational grid of the two hybridization schemes is described. This chapter also contains all the details of the implementation. The frameworks ParadisEO, BOB++ and the new hybridization framework ParadisEO-HEM are presented. The underlaying support of the experiments, the experimental grid Grid'5000, is also presented and the chapter is ended by some results obtained on standard Q3AP benchmarks.

Part I

Combinatorial optimization for permutation-based problems

Chapter 2

Introduction to combinatorial optimization

Contents

2.1 Introduction

Combinatorial optimization (CO) is a branch of applied and discrete mathematics. It consists in finding optimal configuration(s) among a finite set of possible configurations (or solutions) of a given combinatorial optimization problem (COP). The set of all possible solutions noted S is called solution space or search space. COPs are generally defined as follows (Definition 1) [11].

Definition 1 (Combinatorial optimization problems)**.**
A combinatorial problem P=(S,f) can be defined by:
– *a set of decision variables X*
– *an objective function f to optimize (minimize or maximize): $f(x)$*
– *subject to constraints on the decision variables*

COPs are generally formulated as mixed integer programming problems (MIPS) and most of them are NP-hard [42]. According to the required quality of solutions, there are two broad families of search algorithms for COPs, as illustrated in Figure 2.1.

Exact methods aim to find the optimal solution(s) to the problem at hand and prove its optimality. This class of methods includes tree-based algorithms; commonly called branch-and-X methods including branch-and-bound, branch-and-cut and branch-and-price; constraints programming, dynamic programming, A*, etc. Branch-and-X methods are tree-based and enumerative methods which intelligently (or implicitly) explore the whole search space in order to find the optimal solution for the problem. The problem is solved by subdividing it into simpler subproblems.

The second big family of search methods in CO is approximate methods. As a general definition, an approximate search method aims to find a good quality or a near-optimal solution to the tackled problem in a reasonable time by exploring a selected part of the solution space in which good quality solutions are expected. Unlike exact methods, there is no guarantee of optimality of the found solutions. This family of search methods is also composed of different kind of algorithms classified according to their search strategy: approximation algorithms and heuristic algorithms. The most popular search algorithms in the class of heuristic methods are metaheuristics. Metaheuristics are general-purpose optimization methods that can be applied to any kind of problems as they do not contain any problem-specific knowledge in their design line.

Although optimization algorithms are often classified on distinct categories, a recent trend of research in the field consists in mixing or *hybridizing* different kind of algorithms in order to circumvent the limits of each method and solve larger instances of complex optimization problems. Combining exact search algorithms with metaheuristics is one of those new tendencies in the optimization community, as the two families of algorithms offer divergent and complementary characteristics. In this chapter, we will study a tree-based search exact algorithm, the branch-and-bound algorithm and some metaheuristics. Some classifications and taxonomies for hybrid search algorithms are also presented with examples from the state-of-the-art. Finally, such complex methods are time consuming for large problems. Thus, parallelization strategies for both metaheuristics and tree-based exact methods are examined. The chapter ends with some concluding remarks.

2.2 The sequential branch-and-bound algorithm (B&B)

The B&B algorithm [65] is based on an implicit enumeration of all the solutions of the considered problem. The search space is explored by dynamically building a tree whose root node represents the problem being solved and its whole associated search space. The leaves

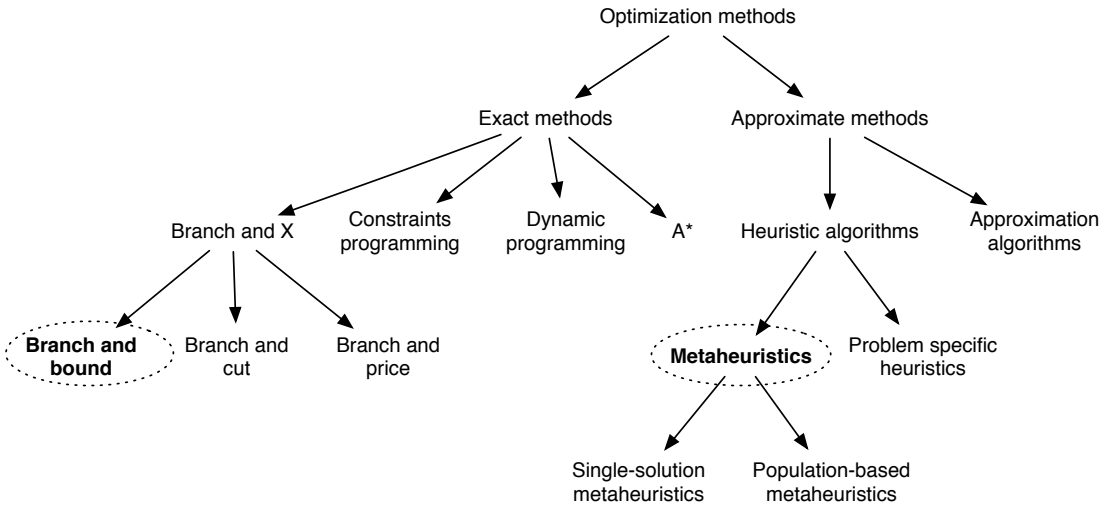


Figure 2.1: Classification of optimization methods (from [102]).

are the potential solutions and the internal nodes are subproblems of the total solution space. The construction of such a tree and its exploration are performed as follows. A global variable is used to store the best solution found so far, it is initialized to ∞ (minimization problem). This value is also called an *"upper-bound"* for the minimization problem. A priority queue is maintained by the algorithm and stores the set of yet unexplored subproblems and initially stores the root node. Among the nodes in the tree, active nodes are those which are in the queue but are not yet decomposed. Three main operators are used in the B&B algorithm.

- **Selection operator:**
  This operator is responsible of selecting the next node to be explored from the queue of active nodes. Possible selection strategies include depth-first, breadth-first and best-first. Depth first strategies starts by exploring one branch until it reaches the leaves or the branch is eliminated. This strategy aims to reach a feasible solution to the problem as soon as possible in order to update the upper-bound. Breath-first strategy consists of exploring all the nodes of the level l before going to the level $l+1$. The main advantage of this strategy is that all the regions of the tree will be considered in the same time in each level. Best-first strategy uses the bounding function to order the set of active nodes according to their bound and select the node with the best bound (which is likely to contain an improving solution). The advantage of this strategy is the possible early improvement of the upper-bound. Indeed, the better is the upper-bound, the faster will be the algorithm because of the pruning operations (elimination of poor subspaces).

- **Branching operator:**
  This operator decomposes a given problem into smaller subproblems. Concretely, at each level in the tree, a different decision variable is assigned. There are two different strategies when decomposing a problem. Dichotomous decomposition and polyatomic

decomposition. In dichotomous decomposition, at each level of the tree, a single $0-1$ decision variable x_i is chosen for branching and two values are possible: $x_i = 0$ or $x_i = 1$. The obtained tree is binary. Alternatively, in polyatomic decomposition, at each level of the tree, a single decision variable is considered and subproblems are constructed by assigning all the possible real values of the variable. In permutation problems, the obtained tree is a permutation tree: at each level l we still have $n - l$ possible values to place in the permutation of size n. For permutation problems, the most used branching strategy is the polyatomic: QAP and Q3AP [58], permutation flowshop problem [76].

– **Bounding:**
A bounding function is used to estimate the quality of the solutions covered by the evaluated node. Some constraints relaxation techniques are used to calculate this estimation, such as Lagrangian relaxation. For example, in [58] a linearization technique is firstly used to reformulate the original problem to solve. Then, the constraints of the problem are eliminated and added to the evaluation function, this is called Lagrangian relaxation. The obtained problem is called dual problem and the original problem is the primal problem. The dual problem is a linear problem that can be solved in a polynomial time using standard LP (linear programming) techniques. The solution found for the dual problem is also a solution for the primal problem if it is feasible. Otherwise, it is a bound for the primal problem. In case of a minimization problem, the bounding function is also called *lower-bound* by opposition to the *upper-bound* which represents the value of the best found solution for the problem. The quality of a bound is measured by the error relatively to the optimal solution of the problem. Tight bounds help the B&B to eliminate a big portion of the solution space that are not likely to contain an improving solution. Thus, this function is the most important function in B&B algorithms.

At each iteration of the algorithm a subproblem is selected according the selection criterion (lower-bound of the subproblem, depth of the subproblem, etc.) from the queue of yet unexplored subproblems. Then, the branching operation is applied on the selected subproblem, subdividing the subproblem's solution space into two or more subspaces to be investigated in a subsequent iteration. For each one of these, it is checked whether the subspace consists of a single solution, in which case it is compared to the best solution found so far and the best one is kept. Otherwise, the bounding function for the subspace is calculated and compared to the fitness of the best solution (the upper-bound). If the bound of the subspace is higher than the fitness of the best solution which means that all the solutions contained in this subspace have higher costs than the best solution found so far, the whole subspace is discarded (pruning or elimination operation). Otherwise, the subspace and its bound are stored in the queue of yet unexplored subproblems. This process is repeated until the queue of active nodes is empty. An illustration of the B&B algorithm is depicted in Figure 2.2.

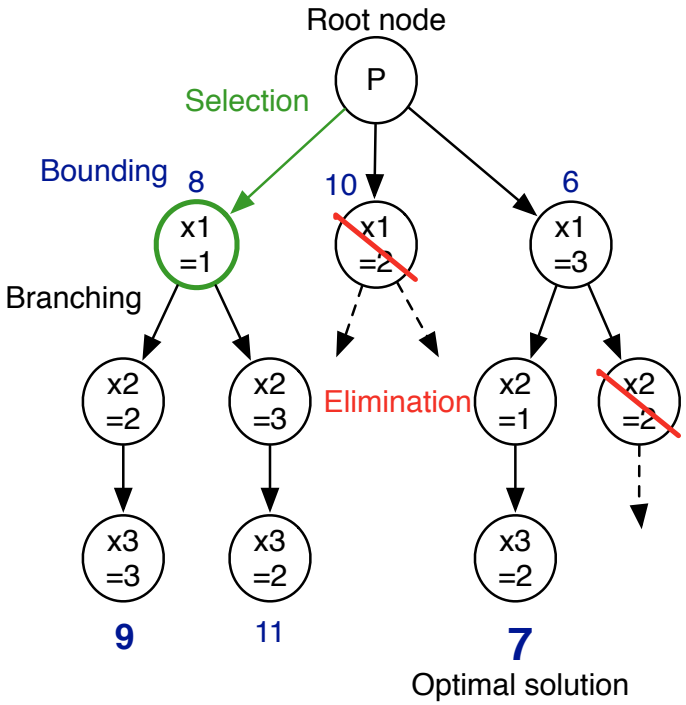


Figure 2.2: Illustration of the sequential branch-and-bound algorithm.

2.3 Metaheuristics

Among approximate methods, metaheuristics are the most popular heuristic search methods as they are general purpose search algorithms that can be applied to solve almost any optimization problem. There is not a unique definition for metaheuristics. Here we quote two definitions by Crainic and Toulouse [38] and Osman and Laporte [80].

In [38], metaheuristics are defined as follows: *"Metaheuristics for optimization problems may be described summarily as a "walk through neighborhoods, a search trajectory through the solution domain of the problem at hand".*

The definition given by Osman and Laporte in [80] is: *"A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."*

Metaheuristics could be single-solution based if they use a single starting point (e.g. local search, simulated annealing, iterated local search and tabu search) or population-based if a population of search points are used (e.g. particle swarm, evolutionary algorithms, ant colony optimization), and many of them are inspired from natural processes (e.g.

evolutionary algorithms from biology and simulated annealing from physics).

In designing of any metaheuristic, two contradictory criteria must be taken into account: exploration of the search space (diversification) and exploitation of the best found solutions (intensification). Exploration means the diversification of the search to different regions in the search space for a better sampling of the solution space. Alternatively, exploitation means the intensification of the search around some good quality solutions in order to find an improving solution. In the same subject, we quote Crainic and Toulouse in [38]: *"Well-designed metaheuristics avoid getting trapped in local optima or sequences of visited solutions (cycling) and provide reasonable assurance that the search has not overlooked promising regions"*. To summarize, a balance should be found between these two contradictory objectives.

In this section, we complete the definition of metaheuristics by presenting some common concepts such as representation (or encoding) of the solutions, neighborhood structures and move operators.

2.3.1 Common concepts of metaheuristics

Some common concepts of metaheuristics are presented in this section. The concepts of representation and neighborhood are defined.

2.3.1.1 Representations and locality of a representation

Central to the design of any search algorithm, is the representation of the problem being solved. A representation associates an encoding, that can be easily manipulated by the search algorithm, to each candidate solution of the problem. For example, if we consider the traveling salesman problem (TSP), a solution of this problem is a tour in which all the cities are listed in the order they are visited and each city is visited only once. This solution (or tour) can be represented using different encodings: binary, graphs, permutations etc. In the permutation representation, the permutation encode the feasible tour as a sequence of cities in which the first and the last elements are connected.

As representation is a key element which determines the efficiency of a metaheuristic, there are some properties a good representation should fulfill [102],[91]:

- **Completeness:** That is all the solutions must have a representation. This property is important because the search operators in metaheuristics operate in the encoded space instead of the effective solution space of the problem. Thus, all the solutions should be represented in order not to be discarded from the search.

- **Locality:** This property is defined relatively to the underlying search operator which manipulates the representation. Locality means neighboring solutions in the encoded space correspond to neighboring solutions in the solution space. Or in other words, a small modification in the encoded solution should correspond to a small change in the real solution of the problem.

– **Efficiency:** The representation must be easily manipulated by the search operators in order to reduce the complexity of the latters. This is essential to the overall complexity of the algorithm and its efficiency.

Relative to the representation is the mapping function which associates every element in the encoded space (genotype) to its corresponding image in the solution space (phenotype). The locality of a representation as defined by Rothlauf in [91] is given in Definition 2.

Definition 2 (Locality)**.**
The locality of a representation represents how well genotype neighbors correspond to phenotype neighbors. The locality of a representation is high if genotype neighbors correspond to phenotype neighbors. In contrast, the locality of a representation is low if some neighboring genotypes do not correspond to neighboring phenotypes.

If a representation with low locality is used, the search in the genotype space will be equivalent to a random search in the phenotype space, which leads to inefficient search (see Figure 2.3 for an illustration). In [91], Rothlauf argues that the locality of a representation can even influence the problem difficulty!

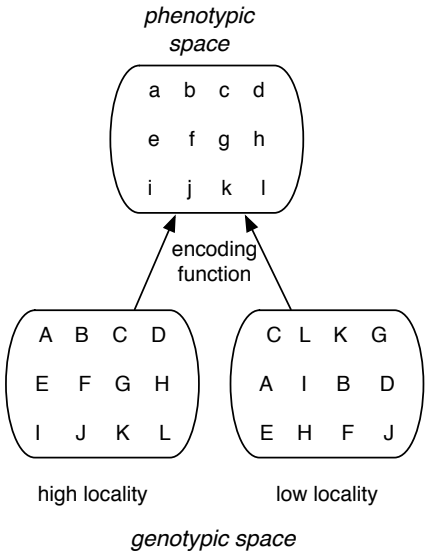


Figure 2.3: Illustration of low locality and high locality representations (from[91]).

2.3.1.2 Neighborhoods and connectivity of neighborhoods

Generally, a *neighborhood structure* is defined as follows (Definition 3):

Definition 3 (Neighborhood structure)**.**
A neighborhood structure is a function $N : S \rightarrow 2^S$ that assigns to each solution s of S a set of solutions $N(s) \subset S$.

A neighborhood structure is constructed by iteratively applying a *move* on the starting solution. A move is defined as a small change or perturbation in the structure of the current solution. Move operators are used in some metaheuristics in order to form a neighborhood structure around the current solution. All or a part of the solutions in this neighborhood are examined with a hope to find an improving solution for the problem. If we consider an optimization problem in which the solutions are represented by permutations, a basic move could be a swapping function which permute the positions of two elements in the permutation. Another notion related to neighborhoods and moves is the *distance* measure.

The distance between two solutions relatively to a given move operator is defined as the number of times the considered operator have to be iteratively applied on the first solution to reach the second solution. Thus, the distance between two neighboring solutions is minimal. Another more detailed definition of neighborhoods is given in Definition 4 [102]:

Definition 4 (Neighborhoods in COPs)**.**
In discrete optimization, the neighborhood $N(s)$ of a solution s is represented by the set $\{s'/d(s', s) \leq \epsilon$, where d represents a given distance related to the move operator.

Among the main properties that a neighborhood structure should have, is the connectivity. The connectivity of a neighborhood $N(s)$ is defined as the existing of a path from any solution in $N(s)$ to any other solution in $N(s)$. In [67], Yong Li et al. introduced the concept of degree of connectivity (C_N) of a neighborhood. The definition of C_N as stated by the authors in [67] is given in Definition 5.

Definition 5 (Degree of connectivity of the neighborhood)**.**
For a given problem P with the solution space represented by $S(P)$, the degree of connectivity of a neighborhood structure N is defined as

$$C_N = 1/K$$

where

$$K = min_{s \in S(P)} min_{p,q \in N(s)} max_{P_0 = P_k = q, i = 1, \ldots k-1} min \;\; d(p_i, p_{i+1})$$

In Definition 5, K is a constant which can be interpreted as follows. Among all possible solutions in the solution space $S(P)$, let's consider each pair of solutions p, q in $N(s)$ and calculate the paths between the pairs. Then, taking the maximum among all such paths, we determine the smallest distance between consecutive solutions in a path between p and q, this defines the constant K.

The concept of connectivity of a neighborhood is tightly related to the distance between the solutions of the neighborhood and the used move operator. In the case of permutation-based problems, let's consider the swap operator and the Hamming distance between permutations. This distance is defined as the number of positions which are different in the two permutations. The minimal Hamming distance between two permutations is equal to $d_{min} = 2$. Since the path between permutations is constructed by a set of transpositions (swaps), which guarantee that the distance between every consecutive elements in the path

is minimal and it exists always a path between any pair of permutations $p, q \in N(s)$. Thus, $K = 2$ and $C_n = 1/K = 0.5$. The neighborhood generated by the swap operator in the space of permutations is highly connected.

2.3.2 Hill Climbing algorithm (HC)

This very basic metaheuristic algorithm, also called descent or simply local search is based on two elementary components. The generation of neighborhood structures using an elementary move function and a selection function which determines which solution in the current neighborhood will be replacing the actual search point. The search continue as long as there is improvement and stops when there is no better solution in the current neighborhood (see Algorithm 1).

Algorithm 1: Template algorithm of a Hill Climbing algorithm

```
1  //Initialization
2  Generate Initial solution s_0
3  s ← s_0
4  //Main loop
5  while not stopping condition reached do
6      Generate(N(s))
7      Evaluate(N(s))
8      if no better neighbor is found then
9          Stop
10     //Select a better neighbor s' ∈ N(s)
11     s ← s'
12 return Local optima
```

The performance of this algorithm depends both on the size of the visited neighborhood and on the selection strategy. The neighborhood size is determined by the used move operator. For example, if we consider a permutation problem of size n and a swap operator simply exchanging the positions of two elements in the permutation, the size of the neighborhood is equal to $n(n-1)/2$. The larger is the size of the neighborhood, the better is the search. Nevertheless, for some problems when the evaluation of one neighbor is time consuming, large neighborhoods are in the contrary, restricted by generating only a partial neighborhood.

The selection strategy determines which direction the search will take by selecting an improving neighbor. There are at least three strategies for selecting the next searching point.

- **Best improvement or steepest descent:** As its name indicates it, in this strategy the neighborhood is evaluated entirely before taking a decision in order to select the best available neighbor in $N(s)$. This strategy aims to go straight to the best solutions in each neighborhood. However, this could be time consuming for large neighborhoods and the search is deterministic, which means we always obtain the same final solution when starting from the same initial solution.

– **First improvement:** Unlike the previous one, this strategy simply selects the first neighbor which improves the actual best found solution.

– **Random selection:** In order to avoid deterministic search, this strategy uses randomness to select one of the improving neighbors.

This algorithm is used to find a local optimum solution in a very short time. Thus, it is not efficient for global optimization especially for large search spaces as it stops when it finds a local optimum.

2.3.3 Iterated local search (ILS)

The template algorithm of the ILS algorithm is illustrated in Algorithm 2. This algorithm was initially designed to circumvent the problem encountered in the local search algorithm (HC): how to escape from local optima. A perturbation function is applied on the current solution and the local search step is repeated from the new starting solution until the stopping conditions are reached. This algorithm is initially formalized and generalized by Stutzle in [99] and in [49]. Three main components can be distinguished from the template algorithm.

– **Local search function:** A basic local search algorithm is used to reach the local optima of the current neighborhood.

– **Perturbation operator:** The objective of this function is to slightly modify the actual local optima in order to reach other neighborhoods which may contain improving solutions. In order to avoid cycling, the perturbation operator should be different than the move operator used in the local search step.

– **Acceptance criteria:**
After the perturbation step, a new solution is obtained and the local search step is repeated staring from this solution. The best found solution in this iteration is compared to the previous best solution in order to decide if the search will continue from the new solution or go back to the previous one. Different strategies of acceptance are possible. In [49], the new optimum is directly accepted for perturbation if its fitness is better than the current best solution. Otherwise, it is accepted with a probability defined in Equation (4.10) if its fitness deteriorates the fitness of the old solution by ω % .

$$accept(s', s) = \begin{cases} fitness(s') > fitness(s) \ \ or \\ fitness(s') - \omega * fitness(s) > fitness(s) \end{cases} \tag{2.1}$$

2.3.4 Genetic algorithms (GAs)

Genetic Algorithms (GAs) are powerful stochastic search and optimization techniques inspired from the theory of evolution and the adaptation of species. They are one of the most studied class of Evolutionary algorithms (EAs). The origin of this algorithm is due to Holland in [51] and [50]. Since then, this algorithm has been used to solve various kind of

Algorithm 2: Pseudo code of the ILS algorithm

Input: Randomly generated initial solution s_0

```
/* Run a local search starting from s_0 */
s_* ← LocalSearch(s_0)
repeat
    /* Perturbation of the actual best solution */
    s' ← Perturbation(s*)
    /* Apply local search on the new solution */
    s'_* ← LocalSearch(s)
    /* Acceptance test */
    s_* ← Accept(s_*, s'_*)
until stopping criterion reached;
return s_*
```

problems [28], [45] and many extensions [60],[4],[108] theoretical and experimental studies have been performed [44], [56], [20].

GAs use techniques inspired by natural evolution such as inheritance, mutation, selection, and crossover to generate better solutions through generations. Algorithm 3 illustrates a typical GA. In the first step, a population of initial search points called "individuals" is generated. Each individual is a coding or a representation (binary, integers, trees, etc.) for a candidate solution to the problem at hand. By analogy to the evolution theory in biology, an individual is composed of one or several *chromosomes* (depends on the structure of the solution). In each chromosome, the *genes* are the variables of the problem and the possible values are referred to as *alleles*. The encoding function associates to each phenotype (the solution of the real problem) a genotype representation. The genotype is used during the reproduction step of the algorithm while the phenotype is necessary for the evaluation of the cost of the individual.

An evaluation function is used to associate a "fitness" measure to each solution indicating its quality. During the evolution, according to the Darwinian principle of the survival of the fittest, the fittest individuals are selected for reproduction according to different selection strategies (e.g. elitism, roulette wheel selection). Then, reproduction (or transformation) operators (crossover and mutation) are applied on each couple of selected individuals to generate new individuals (offspring). Replacement strategies are used to produce the next generation of fitter individuals (e.g. Stochastic tournament replacement). This process is repeated until the stopping criterion is fulfilled (e.g. after m generations or a given time limit is reached).

Each one of the basic components of a GA is detailed in the following.

Algorithm 3: Template of a genetic algorithm

```
1 Generate initial population: P(0)
2 t ⇐ 0
3 while stop conditions not reached do
4 |   Evaluate (P(t))
5 |   P'(t) ⇐ Selection(P(t))
6 |   P'(t) ⇐ ReproductionOps(P'(t))
7 |   P(t+1) ⇐ Replacement(P'(t), P(t))
8 |   t ⇐ t+1
9 Return final population
```

Initialization of initial populations

The initial population of GAs is usually created using pseudo-random generation. Nevertheless, recently it has been pointed out in numerous works [46], [97], [52], [96] that both quality and diversification degree of the initial population can affect convergence and final solution quality in GAs. In fact, the quality of the starting solutions may contribute to get fittest individuals more quickly. Heuristics are also used to get optimized initial populations [52]. Nevertheless, in this kind of initialization strategies, one should use an additional mechanism that prevent the population from being in the same region of the search space. For example, a part of the population could be randomly generated in order to introduce some diversity while the rest of the population is optimized. To conclude, when choosing an initialization strategy, one may take into account the quality of the solutions, their distribution in the solution space and the computational cost of the method used to generate this population.

Selection strategies

The selection mechanism in GAs allows to determine which individuals in the population will be used to produce the next generation. The intuitive selection method would be to select the fittest parents because they are more likely to engender good quality offspring. Variety of selection methods are proposed in the literature and some of them are based on the fitness of the individuals. On the other hand, such an extreme selection may discard poor quality individuals from the population and lead to a premature convergence of the algorithm. Thus, other selection strategies based both on the fitness of the individuals and on some stochastic mechanisms are developed. Examples of the most used selection strategies are: the roulette wheel selection and the rank-based selection.

Crossover operators

A crossover is generally a binary operator which combines two selected individuals (parents) in order to likely produce better offspring. An example of crossover operator for permutation-based problems is the two-points partially mapped crossover (PMX) [44]. This operator has been designed initially for the traveling salesman problem (TSP). Two randomly

chosen points in the permutation are used as cutting points. Each parent is used once as first parent and once as a second parent. Two offspring are formed by permuting the two parents. The components between the two cutting points in the permutation are inherited from the first parent and the remaining positions are filled from the second parent as long as the solution remains feasible (no redundant elements in the permutations). Conflicting positions are filled from the unused elements. Note that in such a situation, what is called an *implicit mutation* occurs. Mutation being the introduction of genes that are not available in the genetic material of the two parents. As a last remark, the crossover operator as well as the mutation operator should be designed with respect to the properties of the tackled problem.

Mutation operators

The mutation operator is a unary operator which is used to introduce a small perturbation on a given individual. Mutation is used as a diversity mechanism in GAs, in order to introduce new genetic material which is not present in the pool of population. In the context of permutation representations, a mutation could consists in simply swapping the positions of two elements in the permutation. In order to avoid a randomization of the search, this operator is used with a small probability p_m generally in the interval $[0.001, 0.01]$. However, in some hybrid approaches, this operator plays the role of an intensification tool, thus higher values could be allowed.

Replacement strategies

The replacement operation consists in deciding which individuals will be admitted in the next generation among both the parents from the old generation and the offspring obtained from the reproduction step. We may distinguish two main strategies.

- **Generational replacement:** This strategy consists in replacing all the parents by their offspring to form the next generation. The GAs using this replacement method are called *generational* GAs.
- **Steady-state replacement:** Here only one offspring is generated for each couple of parents and it replaces the worst parent in the next generation, which forms steady-state GAs.

Those two strategies are extreme. Thus, a variety of other strategies are developed. For example, instead of admitting all the offspring, one may admit a given percentage of them which will replace the worst parents.

2.4 Hybrid optimization algorithms

The complementary behaviors (exploitation/exploration, balance between execution time and quality of the final solutions, etc.) of the methods previously introduced makes hybrid methods combining different types of optimization methods more efficient especially for large benchmarks. Different combinations of optimization methods have been used in the optimization community: exact methods combined to other exact methods, exact methods combined to metaheuristics and/or domain specific heuristics, etc.

This class of methods has been largely studied in the literature. A recent review of the most important works in this domain could be found in [87].

In this section, we present some classifications and taxonomies for hybrid search algorithms and a related work section dealing with the special case of cooperative strategies between GAs and the B&B algorithm.

2.4.1 Taxonomies and classifications

In order to define the new class of hybrid search algorithms, several classifications and taxonomies are proposed in the literature [102],[89]. In this section, we will first introduce two taxonomies for hybrid metaheuristics followed by a taxonomy for hybrids combining metaheuristics with exact search algorithms.

2.4.1.1 Taxonomies for hybrid metaheuristics

In this section we will present two main classifications for hybrid metaheuristics according to different criteria: the classification proposed by Talbi in [102] and the classification proposed by Raidl in [89].

Talbi's classification for hybrid metaheuristics

In this classification, hybridization of metaheuristics is studied from two points of view: design and implementation. From the design point of view, a hierarchical classification in which four main classes are identified, is depicted in Figure 2.4. In the first level of the hierarchical classification, the two classes low level and high level hybrids refer to the functional composition of the hybridized metaheuristics. In low level hybrids, one of the search algorithms is encapsulated by the second method, while in high level hybrids each method is self contained and is viewed as a black box by the other method. The second level of the hierarchical classification dresses the way the hybridized metaheuristics interact with each other, or the way they are executed. The Relay class means the metaheuristics are executed sequentially or in a pipeline fashion, which means the output of one method is the input for the next method in the queue. The four basic classes are then combined to form the following generic classes: Low level Relay Hybrid (LRH), Low level Team-work Hybrid (LTH), High level Relay Hybrid (HRH) and High level Team-work Hybrid (HTH).

1. **LRH (low-level relay hybrid)**
   In this class, a functional component in one method is replaced by the other method and the two methods are executed in sequence. For example, this kind of hybridization can be found in a single-solution metaheuristic embedding another single-solution metaheuristic algorithm in order to intensify the search in some regions at the end of the search. The two methods are executed sequentially. Nevertheless, due to the similarity between the two hybridized metaheuristics, this class of hybrids is not very used.

2. **LTH (low-level teamwork hybrid)**

The LTH scheme regroups hybrid methods combining metaheuristics having divergent and complementary objectives. One of the metaheuristics is embedded in the other one and each metaheuristic is powerful either in exploration of new subspaces or in exploitation of good quality solutions. For example, population-based metaheuristics such as GAs are known to be powerful in exploration because of the use of multiple starting points leading to a better sampling of the solution space. On the other hand, single-solution metaheuristics are known for their exploitation capabilities. The combination of these two classes of metaheuristics justifies by definition the creation of the new class of hybrid metaheuristics whose the objective is to take advantage from the points of strength of each method. For example, it has been shown in many works that hybrid GAs combined with local search (or any s-metaheuristic such as TS and SA) in place of mutation or crossover operators yields much better results compared to standard GAs for combinatorial optimization and real world problems. Yet, this kind of hybridizations is very common in the literature [31], [28].

3. **HRH (high-level relay hybrid)**
   In the HRH schemes, the hybridized metaheuristics are self-contained and executed sequentially. This class contains hybrids in which one of the methods is used either to generate initial solutions for the second method or to improve its final solution. For example, for some problems, greedy heuristics are used to construct a good quality feasible solution and this solution is used as starting solution in the metaheuristic. The other possibility is to use one metaheuristic to globally optimize the problem and use a different metaheuristic to optimize locally around the final best found solution in the global optimization step. This last scheme is used for example to improve the best solutions in the final population of a GA using a powerful intensifying metaheuristic such as simulated annealing (SA) and tabu search (TS).

4. **HTH (high-level teamwork hybrid)**
   This class of hybrids is also considered as a parallel metaheuristic. It involves several self-contained metaheuristics which cooperate together to solve the same problem. Each metaheuristic evolves independently from the others and exchanges information with neighboring metaheuristics following a given interconnection structure or topology. For example, a set of cooperating GAs could be considered as a HTH scheme. Each GA is evolving a different subpopulation and communicates through a given topology with neighboring GAs in order to exchange some individuals to diversify the search. The search operators and parameters used in each GA could be either the same or different. This example is also known as GAs island parallel model in parallel metaheuristics classifications [4]. In the island model even if the cooperating metaheuristics are homogeneous (multiple processes of the same algorithm), it is considered as a hybrid scheme because each island is performing an independent search in a different region of the solution space. In the taxonomy proposed in [102] there is a flat portion which refers to this last consideration (homogeneous and heterogeneous hybridizations).

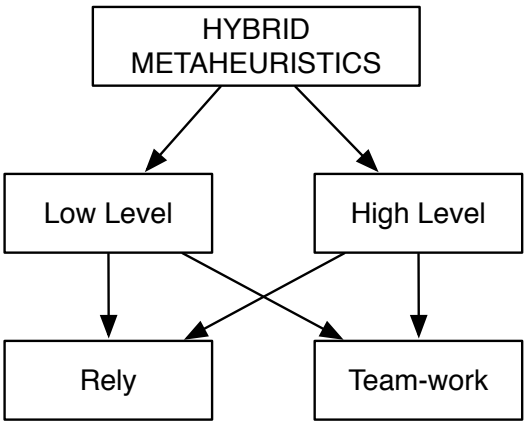


Figure 2.4: Taxonomy of hybrid metaheuristics [102].

Raidl's classification for hybrid metaheuristics

In Raidl's classification described in [89], the hybridization strategies for metaheuristics are simply classified around the following design questions: What is hybridized? What is the level of hybridization? What is the order of execution? What is the control strategy?

1. **What is hybridized?**
   This question allows to distinguish between hybridization strategies in which we combine different metaheuristics, metaheuristics with problem-specific heuristics, metaheuristics with exact algorithms, etc.

2. **What is the level of hybridization?**
   This question is the same as the one considered in Talbi's taxonomy [102]. The level of hybridization can be low level if the metaheuristics exchanges components (not self-contained) or high level if each method is self-contained. These two categories are also referred to as strong coupling and weak coupling respectively.

3. **What is the order of execution?**
   The order of execution can be sequential, interleaved or parallel. Sequential execution is equivalent to the relay class in Talbi's taxonomy. Interleaved execution means the hybridized methods are executed in a round-robin fashion. In parallel executions, additional criteria are considered. They are related to the used architecture, the granularity of the hybrid algorithm, the nature of the hardware (homogeneous/heterogeneous), the type of memory (shared/distributed) the strategy of data allocation among the different parallel processes (static/dynamic) and the synchronization issue between the combined algorithms (asynchronous/synchronous).

4. **What is the control strategy?**

This last question addresses the issue of the control of the global search. Does one of the methods take the commands over the others to control the search or all the methods are participating and collaborating? In the former case, the strategy is called integrative while in the latter it is a collaborative strategy. In integrative strategies, one can find for example hybrid schemes in which a local search is incorporated in a global search metaheuristic (LTH class). In collaborative strategies, two additional criteria are considered: the homogeneity of the combined algorithms and the nature of the parallelism. In the first criterion, it is distinguished between homogeneous hybridizations in which several instances of the same algorithm are run in a parallel collaborative way (HTH class) and heterogeneous hybridizations where different kind of search algorithms are combined. For the parallelism criterion, parallel hybridizations are also referred to as space decomposition and one may distinguish between implicit space decomposition and explicit decomposition. Implicit decomposition is considered when the parallel search threads are starting from different points generated randomly in the solution space without any kind of explicit partition of the decision variables. Otherwise (in case the decision variables are partitioned among the parallel threads), it is an explicit decomposition. This classification according to the space decomposition criterion is inspired from the classification proposed by El-Abd and Kamel in [29].

2.4.1.2 Taxonomies for hybrids combining metaheuristics with exact search algorithms

Hybrid algorithms combining metaheuristics with exact search algorithms deserve a special care because of the heterogeneity of the two classes of search techniques. Indeed, they have nor the same working principles neither the same objectives. We quote Crainic and Toulouse in [38] : *"Compared to exact search methods, such as branch-and-bound, metaheuristics cannot generally ensure a systematic exploration of the entire solution space. Instead, they attempt to examine only parts thereof where, according to certain criteria, one believes good solutions may be found."* Thus, their combination could produce very powerful and robust search methods.

Although this category of hybrid schemes is covered by the two classifications for hybrid metaheuristics presented in this chapter ([102] and [89]), special classifications are proposed in the literature such as the one proposed by Jourdan *et al.* in [55] and the classification by Puchinger and Raidl in [86]. Those classifications aim to give a deeper understanding for this class of hybrids. The classification proposed in [55] is an extension for Talbi's taxonomy for hybrid metaheuristics. The authors studied several works in the literature dealing with hybrid schemes combining metaheuristics with exact search methods and classified them according to the hierarchical taxonomy of hybrid metaheuristics, as this taxonomy is general enough to fit all types of hybrids. In this section we will present the classification by Puchinguer and Raidl [86].

The classification by Puchinger and Raidl

In [86], a review of the recent works on hybrid methods combining metaheuristics with exact search algorithms is given. The hybridization schemes are classified into two main

classes: *"Collaborative Combinations"* and *"Integrative Combinations"* as illustrated in Figure 2.5. In the class collaborative combinations, one may distinguish between sequential and parallel or intertwined executions. In the class integrative combinations, two subclasses are considered: approaches incorporating exact algorithms in metaheuristics and approaches incorporating metaheuristics in exact algorithms.

1. **Collaborative Combinations:** This class refers to hybrid approaches in which each method is self contained and collaborates with the other methods in order to solve the global problem conjointly. When a sequential execution is used, the search could be started by the metaheuristic and the best found solutions are merged in order to solve to optimality a reduced subspace of the problem using the exact algorithm. Another possibility is to simply use the output of the metaheuristic as an upper-bound in the exact algorithm, this is the simplest scheme for the combination of a metaheuristic with exact methods. In parallel or intertwined executions, the two algorithms evolve simultaneously until the stop criterion of the global search is reached. If the intertwined mode is used, each method is executed during a given period of time (or any other criterion) before it gives back the control to the other method. One may consider that the two methods share the same memory module which allows to integrate the search efforts done by each method. If the search is parallel, this corresponds to a multi-thread search in which each search agent evolves separately and has its own memory. The cooperating agents exchange some information in order to guide each other. In this category, we can find hybridization strategies in which a B&B is used to continuously reduce the search space using pruning operations, while a metaheuristic algorithm is initialized from the pool of unexplored subspaces hold by the B&B. The objectives of such a strategy are (1) avoid eliminated subspaces and (2) diversify the search to unexplored areas.

2. **Integrative Combinations:**
   This class refers to hybridization strategies in which one method is used as an inner procedure in the other method. It is divided onto two subclasses, exact algorithms integrated to metaheuristics and metaheuristics integrated to exact algorithms. In the former case, several possible strategies are proposed in the literature [86]. For example, one may use an exact search algorithm as a local search operator in a metaheuristic in order to explore very large neighborhoods. Another idea consists in using exact search to optimally complete a partial solution produced by a crossover operator in a GA. In the case of metaheuristics integrated to exact algorithms, heuristics could be used to select the next node to branch in a B&B algorithm. Many other integrative strategies are reported in [86] such as the use of metaheuristics to guide the exact search. For example in [34], a B&B algorithm uses a GA in order to locate good quality nodes to solve first. Initially, the B&B is executed alone until a certain level in the tree. Then, information is collected from the B&B tree and used to suggest chromosomes to the GA. If the best solution returned by the GA improves the upper-bound of the B&B, this solution will be grafted onto the B&B tree and if it contains variables that are not yet branched, new nodes are built in the tree and the B&B solves them immediately.

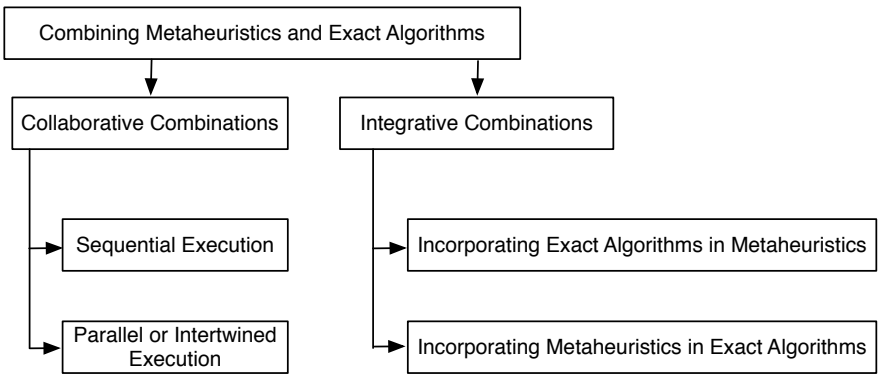


Figure 2.5: Classification for hybrid algorithms combining metaheuristics with exact search algorithms [86].

2.4.2 Related works on cooperative search methods combining GAs with B&B

In order to achieve better computational performances by reducing the number of explored nodes, B&B algorithms are usually combined to other exact and/or heuristic methods ([12], [87], [41], [21], etc). The objectives of such hybridizations differ from one method to the other. In [12], the B&B algorithm is combined with another exact method which is dynamic programming in order to solve the multidimensional knapsack problem (MKP). The used hybridization scheme is cooperative and the resulting method is also an exact method. The B&B is used to solve promising nodes obtained from the previous step of the dynamic programming algorithm. In the rest of the cited works in this section, the B&B algorithm is hybridized with metaheuristics.

Indeed, in this thesis the focus is set on hybrid schemes combining metaheuristics and exact optimization methods. More exactly, hybrid schemes combining GAs and tree-based exact search such as B&B. As already noticed, the simplest way to make a hybrid scheme combining a metaheuristic with B&B is to use the relay scheme. The metaheuristic is firstly executed to get a good approximate solution which will be used as a starting upper-bound for the exact method (B&B). This will help the B&B to reduce the number of explored nodes. Nevertheless, a more judicious exploitation of the two methods could be obtained using a collaborative scheme. This scheme has been used in many works in the literature to solve various problems [21],[41] and [87].

The most important issues in a cooperative search are the determination of:
- The nature of the information to be exchanged between the two methods,
- The use made of the received information in each method,
- The frequency of communications and their nature (synchronous or asynchronous) between the cooperating methods.

Let us focus on the first issue, the nature of information to be exchanged from the B&B to the GA and *vice versa*. This is the key challenge in cooperative methods as it determines

the synergy between the two methods.

In [41], a memetic algorithm is hybridized with a B&B algorithm in a collaborative intertwined may. First, the exact algorithm is executed for a determined number of levels, a set (queue) of promising nodes is constituted. Then, the memetic algorithm is initialized using the pool of B&B nodes. The objective of this initialization is to lead the memetic algorithm to promising regions determined by the B&B algorithm and to improve the upper-bound. In this hybrid scheme, the advantage taken from the collaborative execution is the diversification of the starting populations of the memetic algorithm in order to improve the upper-bound.

The same strategy is used in [21] where a GA is executed in a cooperative way with a B&B algorithm and the initial population of the GA is taken from the list of active nodes managed by the B&B. The goal of this cooperation is to improve the best found solution and to reduce the search space of the GA as the eliminated nodes (and their subspaces) will not be explored by the latter. This means the GA's search should be maintained inside the subspaces determined by one of the nodes in the B&B pool. Nevertheless, the used transformation operators are not adapted to this type of search. As a solution, the authors used a synchronizing operator whose the role is to verify if the solutions in the GA population are not outside the authorized subspaces. Again, the exchanged information is initial populations from the B&B to the GA and best solution from the GA to the B&B.

In [34], if the best solution returned by the GA improves the upper-bound of the B&B, this solution will be grafted onto the B&B tree and if it contains variables that are not yet branched, new nodes are built in the tree and the B&B solves them immediately.

To summarize, in most of the state-of-the-art works dealing with cooperative hybridization strategies combining GAs with B&B algorithms, the information exchanged from GA to B&B is new best found solutions. From B&B to GA the exchanged information consists either in new populations generated from the pool of nodes hold by the B&B, or promising genes to add to the genetic material of the GA's population. In all cases, the two algorithms continue working each one in its own representation of the search. We believe that the unification of the search space representation in the two classes of methods will help to exchange more detailed information such as subspaces to explore (from B&B to GA) and unsolved promising subtrees to solve first (from GA to B&B). The two algorithms will have the same view on the global search. Thus, the cooperative search could be more accurate and fruitful.

2.5 Parallelization strategies for optimization algorithms

With the development of parallel architectures, parallelism is used as a way to solve larger and more complex combinatorial problems. Parallelism in optimization algorithms is used for diverse reasons. From the traditional objective which is the speed up of the execution time of the sequential algorithm (as in the case of the B&B algorithm) to more complex parallel strategies in which parallelism also aims to diversify the search into different regions (such as in parallel GAs). In the sequel, we will detail parallel strategies used for two classes of

search algorithms: exact algorithms and metaheuristics. In the class of exact algorithms we focus on the B&B algorithm and in metaheuristics we focus on parallel GAs (PGAs).

2.5.1 Parallel B&B algorithms

When applied to large benchmarks, B&B algorithms are very time consuming. Therefore, it is essential to parallelize them in order to reduce the execution time. There exist different classifications for parallel B&B algorithms in the literature.

Gendron and Crainic [43] identified three classes of parallelization strategies for B&B algorithms: parallelism of *type 1* when performing operations on the generated sub-problems in parallel, parallelism of *type 2* when the B&B tree is built by several B&B processes which perform operations on different sub-trees simultaneously and parallelism of *type 3* which consists of generating several B&B trees using different operators (different branching strategies, bounding functions and elimination roles).

In [75], Melab identified the same three parallel models for B&B respectively as: parallel evaluation of the bounds and parallel evaluation of a single bound (type 1), parallel exploration of the tree (type 2) and parallel multi-parametric model (type3). The four classes (type 1 parallel model is decomposed into two subclasses) are further explained in the following:

- **P**arallel evaluation of the bounds: In this model, only one B&B process is launched. The parallelism is used to speed up the evaluation of the bounds of a pool of nodes. Each process evaluates one node as this operation is CPU time intensive for some problems.

- **P**arallel evaluation of a single bound This model is similar to the previous one (parallel evaluation of the bounds) as only one B&B process is used. In this model, a set of processes evaluate in parallel the bound of a single node. Indeed, in some problems, the operation of bounding is too awkward, thus it can also be parallelized.

- **P**arallel exploration of the tree : This model consists in launching several B&B processes where all the processes are similar and simultaneously explore different subproblems of the same tree.

- **P**arallel multi-parametric model: As its name indicates it, this model consists in launching simultaneously several B&B processes which differ either by one or more operators or they are parametrized differently. Each B&B process explores a tree which is not necessarily the same compared to its concurrents. The B&B processes can exchange information such as new upper-bounds.

Among the four models, the third one (parallel exploration of the tree) is the most popular and has been the object of abundant work. Its implementation can also be classified into two categories according to the number of pools of subproblems used in the different B&B processes, as reported in the review by Gendron and Crainic [43]:

– Single work pool: here the parallel B&B processes are mainly implemented on shared-memory architectures where many threads access concurrently the common work pool to retrieve subproblems. For example, one of the parallel models proposed in the BOB++ framework uses POSIX threads in shared-memory architectures [27]. It is also possible to implement this model on distributed-memory architectures by using the master/slave paradigm: the master manages the work pool and sends subproblems to the slaves.

– Multiple work pools: here the parallel B&B processes have several memory locations to store the subproblems to be solved. Several organization schemes are possible: each process has its own work pool, each group of processes share a work pool or each process has its own work pool and there is a global pool shared by all processes. This model is generally implemented for distributed systems. Efficient load balancing mechanisms must be implemented in order to fully take advantage of the computing resources. Several works in the literature deal with this type of parallel B&B [76], [1], [25]. In [76] a farmer-worker approach is used to escape the firewalls problem in a grid environment. The farmer holds a global pool of "jobs" and the workers contact the farmer to get units of job to do (a job is a subspace of the tree). In [1], the implementation paradigm used is the hierarchical master-worker. A single supervisor process controls a number of masters. Each master process controls several workers. The global pool of nodes is hold by the supervisor and the master holds its own pool and nodes are dispatched to the workers in order to be solved. In order to design more scalable parallel B&B implementations, peer-to-peer approaches are also used in some works. For example, the approach used in [25] is a peer-to-peer approach based on a master-worker deployment paradigm enhanced with a mechanism enabling inter-workers communications in order to share best upper bounds. The workers are organized in groups in order to optimize inter-cluster communications.

2.5.2 Parallel metaheuristics

Parallel metaheuristics are studied in this section through the presentation of a state-of-the-art classification proposed by Crainic and Toulouse in [38]. This classification is followed by a section on parallel GAs (PGAs).

2.5.2.1 Classification for parallel metaheuristics

In [38] three generic classes of parallel metaheuristics are defined and studied in the context of three main metaheuristics: genetic algorithms, simulated annealing and tabu search. The parallelization strategies applied to metaheuristics are classified according to the source of parallelism. The three classes of parallelization strategies are depicted in Figure 2.6.

– **Type 1 class** contains parallelization strategies in which the sequential algorithm is not altered. Thus, the explored path is the same in the two algorithms. For example, in GAs the evaluation and transformation steps can be parallelized. This class of parallelism aims to reduce the overall execution time by accelerating a repeated phase

of the sequential algorithm. The performance evaluation in this class could be done using the traditional speed up measures.

– **Type 2 class** contains parallelization strategies in which the source of parallelism comes from the explicit decomposition of the search space by partitioning the set of decision variables into disjoint subsets which are optimized by a set of processors. The same algorithm is executed on each processor but on a different regions of the search space. This class of parallelism aims to avoid redundant search in the parallel implementation and also to diversify the search to different regions in the solution space. Thus, the search path and the final solution quality are different compared to the sequential algorithm.

– **In Type 3 class** multiple search threads concurrently evolve in parallel starting from different initial solutions. Thus, they explore different regions of the search space. The main objectives of this class of parallel strategies are the improvement of the final solution by exploring different parts of the search space, and at a second level, the speed up of the overall execution time. Indeed, the parallel searches are generally executed during a shorter period compared to the sequential algorithm. In some metaheuristics, the total number of iterations in the sequential algorithm is divided by the number of concurrent searches. In cooperative GAs, the number of generations in each cooperating GA is the same as in the sequential version. Instead, the size of the population in the sequential algorithm is divided by the number of GA threads. The multi-search threads may exchange information which means the search is cooperative, or the threads evolve independently from each other. For example, coarse-grained parallel GAs (island model) is a type 3 cooperative parallel algorithm.

Hybrid models can also be created from the three types. For example, one can use type 3 parallelism at the top level and type 1 or type 2 at low level. This will create a kind of hierarchical parallelism.

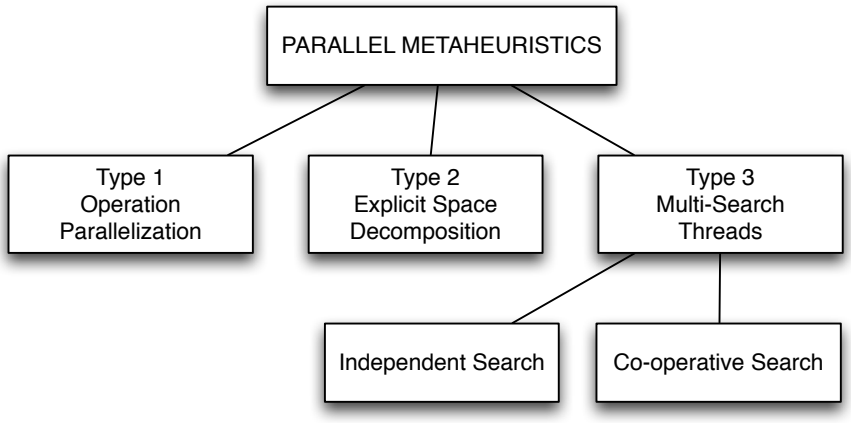


Figure 2.6: Taxonomy of parallel metaheuristics [38].

In the following, we focus on parallel models used for EAs generally and GAs particularly.

2.5.2.2 Parallel GAs (PGAs)

In addition to the traditional speed-up of the execution time, there are several other advantages took from the introduction of parallelism in GAs and in EAs generally. For example, as pointed out in [38], when parallel searches are started from different points in the solution space (populations in the GA), this leads to a diversification of the search and improves the final solution quality compared to the sequential algorithm. A faster execution can also be obtained in PGAs when only one GA is executed and the time consuming tasks are performed in parallel. In [3], Alba studied parallel EAs in this context and argued that some PEAs models can even achieve super-linear performance because of the new features introduced by PEAs compared to their sequential counterparts. Thus, PEAs and PGAs are often considered as a new class of metaheuristics.

From the implementation point of view, due to the use of a population of candidate solutions, GAs are naturally prone to parallelism. For example, Type 1 parallelism (in the classification of parallel metaheuristics) can easily be implemented during the two steps evaluation of the population and transformation. There is not a unique terminology for qualifying parallel models for EAs. In this thesis, we will refer to the classification given by Alba and Tomassini in [4] and in the same time, give equivalent qualifications used in other works for the same models. Another survey of parallel models for GAs and applications by Cantu-Paz and Goldberg can be found in [18] and a more recent work on the same subject by Konfrst can be found in [60].

In [4] parallelization strategies for EAs are classified according to the granularity of the model, which refers to the size and number of populations used in the algorithm. Panmictic EAs refer to the sequential algorithm in which a single population is evolved by a single algorithm. Structured PEAs refer to PEAs in which several structured subpopulations are used instead of one single panmictic population. This class contains two subclasses: fine grain models, also known as cellular EAs (cEAs), and coarse grain models referred to as distributed parallel EAs (dEAs) in [4]. A third class is referred to as global parallelization model. This model is similar to the sequential panmictic EA as it contains a unique population, but some intensive tasks are performed in parallel. A fourth class is considered as the hybridization of different models. When applied to GAs, the instantiation of these four classes is illustrated in Figure 2.7.

Global parallelization or Master/slave models

In master/slave models, a single large panmictic population is used. The behavior of the sequential algorithm is not altered. The evaluation and/or transformation (crossover+mutation) tasks in GAs are distributed among slaves. Selection and population management are still ensured by the master. Because the main objective of this model is to achieve a speed-up compared to the panmictic GA, it is generally implemented for parallel architecture (shared memory machines and distributed systems).

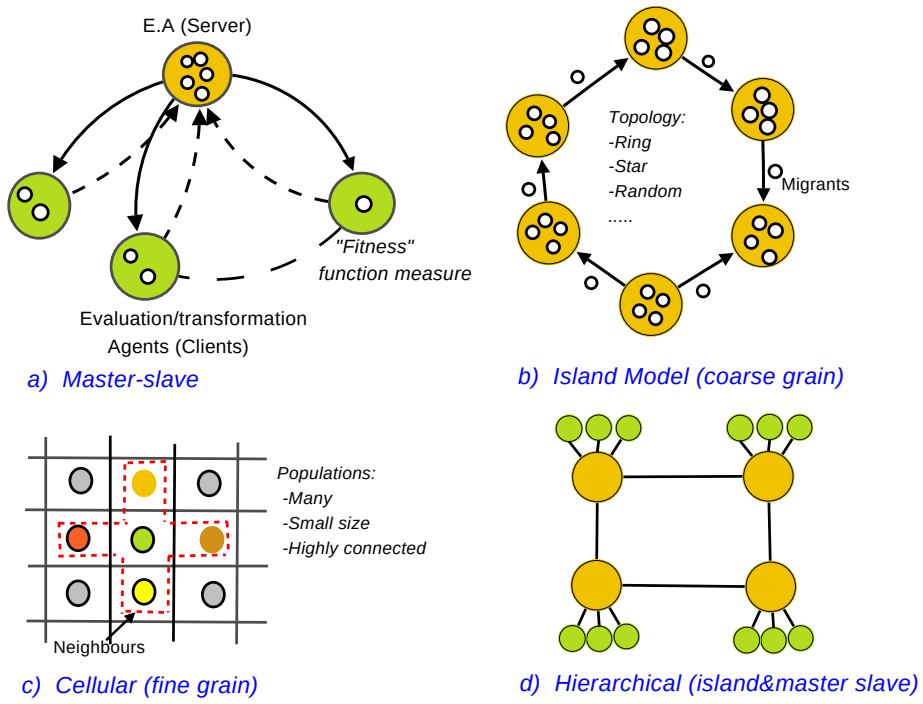


Figure 2.7: Parallel models for evolutionary algorithms [4]

Coarse grain parallelism (Island model)

In coarse grain models several independent GAs are evolving in parallel in distinct subpopulations. This is also known as island model in [107] and as multiple-deme in [18]. Each island is running a panmictic GA on a different subpopulation. In order to circumvent the lack of diversity in each island, a migration operator is added to the standard genetic operators. Indeed, at a given frequency, a given number of individuals is migrated from each island to the neighboring islands. The neighbors are defined following a given topology (ring, star, complete, etc.). The first objective of this model is not necessarily the speed-up but a better coverage of the solution space and the achievement of a better final solution quality thanks to the cooperation between islands. Such objective can be achieved even if the islands are executed on the same machine.

Many studies for island model have been proposed in the literature. The number of populations and their adequate sizes are analyzed by Goldberg *et al.* in [20] and by Whitley *et al.* in [107].

The influence of migration sizes and intervals (migration frequency) is studied by Skolicki and De Jong in [98] and by Alba and Troya in [5]. According to these two studies, the migration frequencies should be moderated in order to give the islands sufficient time for their local evolution. Thus, too frequent migrations may obstruct the evolution of islands, and too big migration interval may lead to less diversity in each island. On the other hand,

the number of individuals to migrate has not to be necessarily big. Indeed, Skolicki and De Jong in [98] argued that big migrations are not necessary and estimated the percentage of individuals to migrate to less than 10% of the size of the population.

The topology is also an important parameter. According to Cantu-Paz and Goldberg in [19], a high connectivity and a small diameter of the topology are necessary to ensure adequate mixing between the different subpopulations.

The communications between neighboring islands in the topology can be synchronous or asynchronous. Alba and Troya analyzed the two communication strategies in [6].

Fine grain parallelism (cellular model)

Fine-grain models commonly known as cellular GAs [108] use a single population which is totally distributed on a connected structure. Indeed, the population is mapped onto a complete connected graph, one individual per node. An edge between two individuals (nodes) represents the possibility for them to reproduce. Thus, unlike the sequential GA, reproduction is allowed only between neighbors. Standard replacement strategies are used to generate the next population.

Hybrid models

Also called hierarchical PGAs, hybrid parallel models are made by the combination of two or more parallel models among the coarse grain, fine grain and master/slave models. A very common hierarchical model consists in using cooperative islands in the first level and each island implements a master/slave model in order to parallelize the evaluation step.

2.6 Conclusion

Combinatorial optimization methods split into two main families: exact and heuristic methods. In this chapter we presented some search algorithms belonging to the two families: the B&B algorithm for exact search and the metaheuristics HC, ILS and GAs. Because of the complementary behaviors of the studied methods, hybridization algorithms emerged as a new efficient class of metaheuristics. State-of-the-art classifications and taxonomies for hybrid optimization algorithms generally including hybrid metaheuristics and hybrid methods combining metaheuristics with exact search algorithms are studied. Such complex methods are time consuming. Thus, parallelization strategies are developed in the literature to take advantage from the development of parallel architectures. A state-of-the-art classification of parallel metaheuristics is also presented as well as an equivalent work on parallel B&B strategies. The chapter gives a general overview on recent advances in the field of parallel and hybrid algorithms for combinatorial optimization. In the next chapter, we will present the target optimization problems we address in this thesis, permutation-based problems.

Chapter 3

Permutation-based problems

Contents

3.1 Introduction

Permutation-based problems in optimization use permutations to represent potential solutions. Applications of permutation problems could be found in different real life areas

such as assignment applications which include a variety of design problems in different areas, scheduling and production sequencing applications, classical traveling salesman problem, routing problems, etc. The mathematical properties of permutations could be useful for search algorithms applied to this kind of problems. The design of neighborhood operators, distance measures between solutions, landscape analyses and other tools could be more accurate if they exploit the concepts of transpositions, inversions, cycles, product of permutations, and so on. In this chapter, we start by introducing permutations as mathematical objects followed by the introduction of some related notions such as transpositions, inversion tables, Lehmer codes and some algorithms for the enumeration of permutations. Then, some optimization problems based on permutations are reviewed in Section 3.3. In Section 3.4, those problems are classified according to the interpretation of the permutation and in Section 3.5 neighborhood and genetic operators for each class of permutation problems are presented. The chapter is closed by some concluding remarks.

3.2 General properties of permutations

In this section, we will review some important properties of permutations that can be useful in the design of search algorithms for permutation-based problems.

3.2.1 Definitions

Permutations are used in different domains of mathematics. Thus, there are multiple definitions according to the use made of them in each domain. In combinatorics, a permutation is an arrangement of a finite set of n objects noted $[n] = \{1, 2, .., n\}$, in a specific order and each element appears only once. There are exactly $n!$ possible permutations on $[n]$.

In group theory (algebra), a permutation can also be seen as a bijection from the set $[n]$ to itself. There is a mapping function from $[n]$ to $[n]$ which associates to each element i a unique image j in $[n]$ and to each image j corresponds one and only one element i. If the permutation is noted π, the mapping function $pos_\pi(i)$ indicates the position of the element i in the permutation π. Alternatively, $\pi(i)$ indicates the element at position i in the permutation π.

There are multiple notations to represent permutations. In this thesis, we use the linear notation which consists in listing the elements a_i in the permutation as follows: $a_1 a_2 \dots a_n$.

The set of all possible permutations of size n designated as S_n, forms a group with the composition of mappings as product and the identity permutation as neutral element. The identity permutation is the permutation $a_1 a_2 .. a_n$ such that: $a_1 < a_2 < .. < a_n$. For example, the identity permutation on the set $\{1, 2, 3, 4, 5\}$ is the permutation 12345. The product of two permutations $\pi_1.\pi_2$ is a permutation π such that

$$\pi(i) = \pi_1(\pi_2(i)) \ \ \forall \ \ i \in [n]$$

For example, if we have $\pi_1 = 3142$ and $\pi_2 = 4213$ then the product $\pi_1\pi_2$ is equal to the permutation $\pi = 2134$. Note that $\pi_1\pi_2 \neq \pi_2\pi_1$.

3.2.2 Transpositions

A transposition is the result of the interchange of two elements in a permutation π. In the rest of this thesis, we will note $\tau_{i,j}(\pi)$ a transposition of the two elements in positions i and j in the permutation π. If the two positions to be interchanged are adjacent, this transposition is named swap.

3.2.3 Order relation for permutation enumeration

There is not a unique way to enumerate the set of all possible permutations of size n. In this thesis, we will focus on two ways, *lexicographic* and *transposition*. The two terms are defined in the following.

Definition 6 (Lexicographic order)**.**
Let π and ψ be two permutations of size n, π and ψ (in this order) are enumerated in a lexicographic order (we note $\pi < \psi$ or π precedes ψ in the enumeration) if

$$\exists i \in \{1, 2, ..., n\} \quad with \quad \pi(i) < \psi(i) \quad and \quad \forall j < i, \quad \pi(j) = \psi(j).$$

For example, the permutations 132 *and* 213 *are enumerated in a lexicographic order.*

Definition 7 (Transposition order)**.**
Let π and ψ be two permutations of size n, π and ψ (in this order) are enumerated in a transposition order if ψ is obtained by applying only one transposition on tow adjacent elements in π. For example, the two permutations 132 *and* 312 *are enumerated in a transposition order.*

3.2.4 Inversions, inversion tables and Lehmer code

According to the definition given in [59], an inversion in a permutation $\pi = a_1 a_2 ... a_n$ is a pair of elements (a_i, a_j) such that $i < j$ and $a_i > a_j$. For example, in the permutation 52143 there are the following inversions

$$(5, 2), (5, 1), (5, 4), (5, 3), (2, 1) \quad and \quad (4, 3). \tag{3.1}$$

The only permutation which does not contain inversions is the identity permutation because all its elements are ordered.

The other concept in relation with inversions is *inversion tables*. The inversion table $T(\pi)$ of a permutation $\pi = a_1 a_2 ... a_n$ is a vector $b_1 b_2 ... b_n$ with b_i the number of elements that are greater than i and appearing to the left of i in the permutation. For example the inversion table of the permutation 52143 is 21210. $b_1 = 2$ because there are two elements (5 and 2) that are greater than 1 and placed at lower positions in the permutation (to the left of 1).

The elements b_i in the inversion table can also be seen as the number of inversions in the permutation in which the element i is the second component. By definition, the elements b_i in the inversion table always satisfy the Condition (3.2).

$$0 \leq b_1 \leq n-1, \quad 0 \leq b_2 \leq n-2, \quad \ldots \quad 0 \leq b_{n-1} \leq 1, b_n = 0 \tag{3.2}$$

Note that each inversion table uniquely indicates the corresponding permutation. There are different methods to determine the corresponding permutation of a given inversion table. Among those methods, in [73] and [82], a data structure that helps to determine the positions of each element is used as explained in Equation (3.3). The complexity of such a method is linear ($O(n\ log\ n)$). A vector N of n integers is used to keep the n elements of the permutation in a numerical order.

$$pos_\pi(i) = N_i[b_i + 1],\ \ with\ \ N_i = N - \{pos_\pi(1), pos_\pi(2), \ldots, pos_\pi(i-1)\} \tag{3.3}$$

The following is an example in which the permutation 52143 is reproduced from its inversion table 21210:

$$\begin{array}{l} pos_\pi(1) = N_1[2+1] = 3,\ \ N_1 = \{1,2,3,4,5\} \rightarrow \pi = *\ \ *\ \ 1\ \ *\ \ * \\ pos_\pi(2) = N_2[1+1] = 2,\ \ N_2 = \{1,2,4,5\} \rightarrow \pi = *\ \ 2\ \ 1\ \ *\ \ * \\ pos_\pi(3) = N_3[2+1] = 5,\ \ N_3 = \{1,4,5\} \rightarrow \pi = *\ \ 2\ \ 1\ \ *\ \ 3 \\ pos_\pi(4) = N_4[1+1] = 4,\ \ N_4 = \{1,4\} \rightarrow \pi = *\ \ 2\ \ 1\ \ 4\ \ 3 \\ pos_\pi(5) = N_5[0+1] = 1,\ \ N_5 = \{1\} \rightarrow \pi = 5\ \ 2\ \ 1\ \ 4\ \ 3 \end{array}$$

Lehmer code

A similar code to inversion tables is the Lehmer code attributed to Derrick Lehmer. Same as inversion tables, the Lehmer code associates a unique code $L(\pi)$ to each permutation $\pi = a_1 a_2 \ldots a_n$. $L(\pi)$ is a sequence $L(\pi) = l_1 l_2 \ldots l_n$ with l_i the number of elements that are smaller than a_i and that appear to the right of a_i in the permutation. The Lehmer code of the permutation 52143 is 41010. Same as in the inversion table, the elements l_i in the Lehmer code always satisfy the Condition (3.4).

$$0 \leq l_1 \leq n-1,\ \ 0 \leq l_2 \leq n-2,\ \ \ldots\ \ 0 \leq l_{n-1} \leq 1, l_n = 0 \tag{3.4}$$

To go back from the Lehmer code to the permutation that generates it, the method used in inversion tables could be used also for Lehmer codes with the difference that in the inversion tables we look for the positions of the elements i in the permutation, while in the Lehmer code we look for the $\pi(i)$ elements as explained in Equation (3.5).

$$\pi(i) = N_i[l_i + 1],\ \ with\ \ N_i = N - \{\pi(1), \pi(2), \ldots, \pi(i-1)\} \tag{3.5}$$

In the following example, we try to reproduce the permutation 52143 from its Lehmer code 41010:

$$\begin{array}{l} \pi(1) = N_1[4+1] = 5,\ \ N_1 = \{1,2,3,4,5\} \rightarrow \pi = 5\ \ *\ \ *\ \ *\ \ * \\ \pi(2) = N_2[1+1] = 2,\ \ N_2 = \{1,2,3,4\} \rightarrow \pi = 5\ \ 2\ \ *\ \ *\ \ * \\ \pi(3) = N_3[0+1] = 1,\ \ N_3 = \{1,3,4\} \rightarrow \pi = 5\ \ 2\ \ 1\ \ *\ \ * \\ \pi(4) = N_4[1+1] = 4,\ \ N_4 = \{3,4\} \rightarrow \pi = 5\ \ 2\ \ 1\ \ 4\ \ * \\ \pi(5) = N_5[0+1] = 3,\ \ N_5 = \{3\} \rightarrow \pi = 5\ \ 2\ \ 1\ \ 4\ \ 3 \end{array}$$

This mapping between the set of permutations and the set of their Lehmer codes or inversion tables could be used to translate any problem represented by permutations into an equivalent problem represented by Lehmer code or inversion tables. This may simplify some of those problems. For example, in group theory and discrete mathematics, finding the

values of left-to-right maxima is simplified if inversion tables are used instead of permutations. Indeed, the values of left-to-right maxima are directly indicated by the positions of the zeros in the inversion table [1]. In the following, we will see how to use Lehemer code to produce permutations in lexicographic order.

3.2.5 Generating the i^{th} permutation in a set S_n

Let S_n be the set of all possible permutations on $[n]$. It is also possible to use the properties of Lehmer code and/or inversion tables to define a one-to-one correspondence between the set S_n and the natural numbers $1, 2, \ldots, n!$ referring to the place of the permutation in the enumeration process. Moreover, if Lehmer code is used, the order produced in the permutations is a lexicographic order.

Factorial number system could be used for this purpose. Factorial number system is a mixed radix numeral system in which the radix of the i^{th} digit from the right is i. Thus, the i^{th} digit from the right has to be less than i and the first one is always equal to 0. For example, the number 22 in decimal base is represented in factorial system by

$$3_3 2_2 0_1 0_0$$

To reconstruct the number expressed in the decimal system from its factorial representation, that we represent simply by the sequence $d_1 d_2 \ldots d_n$, each digit in the factorial number is multiplied by the factorial of its radix, and the sum of all the multiplications gives the number in decimal as illustrated in Equation (3.6).

$$\sum_{i=1}^{n} d_i \cdot (n-i)! \tag{3.6}$$

In the following example, we will try to reconstruct the number 22 from its factorial number $3_3 2_2 0_1 0_0$

$$3 \cdot 3! + 2 \cdot 2! + 0 \cdot 1! + 0 \cdot 0! = 22$$

Each factorial number represents a unique number in the decimal base. The largest number in decimal that we can represent with a factorial number of n digits is $n! - 1$. Until now, we are able to go back from the factorial number to the corresponding decimal number. The next question to answer is how to generate a factorial number from a decimal one. Since the properties of mixed radix systems apply to the factorial number system as well, a number in decimal is converted to its factorial representation by iteratively dividing it by the positions in the factorial sequence (1,2,3,...) and the remainder of the division is the digit of the i^{th} position from the right. The integer quotient is divided in the next iteration until it becomes 0, see Algorithm 4. For more information about factorial number systems please refer to [59].

Now, what is the relation with permutations? The factorial representation of a natural number is nothing but the Lehmer code or the inversion table of a permutation. If the factorial number is considered as Lehmer code, when the latter is converted into a

1. In a given permutation $a_1 a_2 \ldots a_n$, a_i is a left-to-right maximum if $a(i) > a(j) for \ \ all \ \ j < i$

Algorithm 4: Converting a decimal number to factorial number

Input: Decimal number M

Output: Representation of M in factorial system

```
1  i ← 1
2  //First digit from right is always 0
3  d_1 ← 0
4  //Main loop
5  while M ≠ 0 do
6  |   i ← i + 1
7  |   d_i ≡ M  mod i
8  |   M ← M/i
9  n ← i
10 return d_n ... d_1
```

Decimal	Factorial (Lehmer)	Permutation
0	000	123
1	100	132
2	100	213
3	110	231
4	200	312
5	210	321

Table 3.1: Correspondence between permutations of size 3, Lehmer codes and natural numbers

permutation, the order produced is lexicographic.

To summarize, it is possible to generate the i^{th} permutation in a lexicographic order by first converting i to the factorial number system, then the factorial representation of i is interpreted as Lehmer code and converted to the corresponding permutation. In the other way (permutation to number) the Lehmer code of the permutation is calculated and then converted to a natural number. Both coding and decoding operations from permutations to integers and *vice versa* are done in a linear time.

A correspondence between numbers in decimal base, factorial numbers of size 3 (or Lehmer codes) and permutations (size 3 as well) is given in Table 3.1.

3.2.6 Systematic generation of all permutations

To generate all the permutations on a set $[n] = \{1, 2, \ldots, n\}$, one can also use the properties of Lehmer codes and inversion tables. It is sufficient to convert the numbers $1, 2 \ldots, n!$ to their factorial numbers and then to permutations. The order obtained is lexicographic.

Although this method is an elegant way of generating permutations in lexicographic order, it is not the best method in terms of complexity. The fastest algorithm used to generate all the permutations is the algorithm known as the Steinhaus-Johnson-Trotter algorithm or Johnson-Trotter (the authors that independently discovered the algorithm)[103, 54]. This algorithm is the fastest because it generates the next permutation from the actual one by transposing one pair of elements at a time (transposition order). This algorithm is illustrated in Algorithm 5. A mobility direction is associated with each component k in a permutation to indicate if the component to exchange with k is to the left or to the right of k. A mobile integer in the permutation is an integer that is not yet moved to all the positions in the permutation (as a first component in the transpositions). For example, for $n = 3$ the algorithm will generate the permutations in the following order:

$$
\begin{array}{ccc}
< 1 & < 2 & < 3 \\
< 1 & < 3 & < 2 \\
< 3 & < 1 & < 2 \\
3 > & < 2 & < 1 \\
< 2 & 3 > & < 1 \\
< 2 & < 1 & 3 >
\end{array}
$$

Algorithm 5: Johnson-Trotter algorithm for the enumeration of all permutations using transpositions

Input: n
Output: All the permutations on $[n]$

```
//Initialization
Initialize the first permutation by the identity permutation 123...n with all the
mobility to the left, we use the notation < 1 < 2 < 3 ··· < n to indicate the direction
//Main loop
while there exists a mobile integer do
    find the largest mobile integer k
    swap k and the adjacent integer in the direction of k's mobility
    for each integer i do
        if i > k then
            reverse the direction of mobility of i
```

3.2.7 Distance measure between permutations

The notion of distance between permutations is another important notion for metaheuristics working on permutation-based problems. Generally speaking, the distance between two permutations measures how different are the two permutations. It can also be seen as the shortest path between the two permutations or as the number of intermediate permutations obtained by successive small transformations (such as transpositions) from the first permutation until the second permutation is obtained. Generally speaking, Definition 8 gives the

three mathematical properties a distance metric should fulfill.

Definition 8 (The three properties of a distance measure on a set E)**.** *Let E be a set of objects and $x, y \in E \times E$. A distance on the set E, is a mapping function $d : E \times E \to R^+$ such that*
- *$d(x, y) = 0$ if and only if $x = y$ (separative property)*
- *$\forall x, y \in E \times E : d(x, y) = d(y, x)$ (symmetrical property)*
- *$\forall x, y, z \in E \times E \times E : d(x, y) + d(y, z) \geq d(x, z)$ (triangular property)*

One of the most known mathematical distance metrics is the Hamming distance [106]. This distance, was initially defined to measure how two binary strings differ from each other. This measure is applied to error detection and error correction in transmission systems. It counts the number of bits in which the two strings differ from each other. Since then, this distance has been used in many domains and has been generalized for permutations as well. The Hamming distance for permutations is known as *exact match distance* or *generalized Hamming distance* in [90] and is defined as the number of positions in which two permutations differ from each other. Formally:

$$d(\pi, \pi') = \sum_{i=0}^{n-1} x_i \;\; x_i = 1 \;\; if \pi(i) \neq \pi'(i), 0 \;\; otherwise \tag{3.7}$$

3.3 Permutation-based problems in optimization

In this section, some examples of permutation-based problems in combinatorial optimization are summarized and classified by category of general domain of application.

3.3.1 Assignment problems

This category regroups problems consisting in one-to-one assignments between two or more sets of objects of the same cardinality with the optimization (minimization or maximization) of a certain cost related to the domain of application. The complexity of each problem is determined by the constraints of the assignment, the size of the problem (number of elements in the two sets to assign) and the domain-specific cost function to optimize.

The simplest assignment problem known is the linear assignment problem (LAP). Given two sets of objects of the same size n, let it be a set of jobs to execute on a set of machines. Each job has to be assigned to a unique machine and each machine is assigned a unique job. The costs of the assignment of each job to each machine are stored in a matrix $C = C_{(ij)}$, called matrix of costs with C_{ij} the cost of the assignment of the i^{th} job to the j^{th} machine, the problem is to find an assignment p which optimizes the sum of all pair-wise assignments. This problem is not a candidate for hybrid optimization algorithms as it can be solved in a polynomial time using the Hungarian algorithm [63]. Indeed, we are interested only on NP-hard assignment problems. In the following, a non exhaustive list of assignment problems is given. For each problem a brief description is given including origin of the problem, the mathematical and the permutation formulation and the optimization methods used in the

literature to solve the problem. More informations about assignment problems could be found in [15].

3.3.1.1 Quadratic assignment problem (QAP)

Origin and applications of QAP

The quadratic assignment problem is one of the hardest combinatorial optimization problems. This problem was initially introduced by Koopmans and Beckmann in 1957 [61] as a mathematical model for the allocation of indivisible economical activities. More precisely, the problem of assigning plants to locations with the admission of inter-plants transportation costs, which makes the new problem more complex compared to the linear assignment problem (LAP). This application of QAP is also called facility location.

Since then, it has been proved NP-hard by Sahni and Gonzalez[93] and gained a growing attention in the optimization community because of two factors: the astonishing number of applications in different real life domains and its challenging complexity which makes it one of the most known fundamental problems in optimization. Applications of QAP can be found in a disparity of applications such as the design of hospital layouts [30], design of campus layouts [26], typewriter keyboard design in computer manufacturing [84] and the placement of electronic components [88]. A recent review for this problem including different mathematical formulations, applications and a complete state-of-the-art of exact and heuristic methods applied to the QAP is proposed by Loiola *et al.* in [70].

Mathematical formulations for the QAP

In the original problem stated by Koopmans and Beckmann, we are given a set of n plants (or facilities) to assign to a set of n locations knowing the distance relations between the locations stored in a n by n distance matrix $D = (d_{ij})$, and the flow between the facilities (for example, the work relations between different clinics in a hospital), stored in a so called flow matrix $F = (f_{kp})$. The problem is to find a mapping between facilities and locations which minimizes the sum of the pair-wise cost of all the assignments. What makes this problem more complex compared to LAP is that the cost of each assignment between any pair of facility-location, depends not only on the distance and flow between the two elements but also on the cost of the rest of the assignments facility-location.

The so called Koopmans-Beckmann mathematical general formulation of the QAP is given in Equation (3.8). The term d_{kp} indicates the distance between the location k and the location p. The term f_{ij} refers to the flow between the facility i and the facility j. The variables x_{ik} are binary variables which indicate if a facility i is assigned to the location j. If the costs b_{ik} of the assignment of a facility i to a location k is also considered (for example the cost of implementation), the sum of all assignment costs of all the facilities is added to the global cost of a QAP solution. However, this term is generally omitted by most researchers because it is a linear assignment easy to optimize.

Equation (3.9) illustrates the uniqueness constraints of the problem (one facility per

location and vice versa).

$$min \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} \cdot d_{kp} \cdot x_{ik} \cdot x_{jp} \; + \sum_{i,k=1}^{n} b_{ik} x_{ik}, \;\; x \in X \tag{3.8}$$

$$x \in X \equiv \begin{cases} x \in \{0,1\}\,; \\ \sum_{i=1}^{n} x_{ij} = 1,\; j = 1,..,n; \\ \\ \sum_{j=1}^{n} x_{ij} = 1,\; i = 1,..,n \end{cases} \tag{3.9}$$

Since then, another formulation of the problem based on integer linear programming (LP), is given by Lawler [66]. In Lawler's LP formulation, the terms $(f_{ij} \cdot d_{kp})$ are replaced by a unique term c_{ijkp} which represents the "cost" of the assignment of facility i to location k and facility j to location p. Those costs are stored in a so called cost matrix C of size $n^2 \times n^2$. In this thesis we are more interested on the permutation representation of the problem. Indeed, QAP can be formulated as the problem of assigning n facilities to n adjacent locations while minimizing the cost of the global assignment. Thus, a solution for the QAP can be represented as a permutation π of n numbers $\{1, 2, \ldots, n\}$ representing the facilities and the positions in the permutation represent the n locations. This formulation is given in Equation (3.10).

$$min \sum_{i,j=1}^{n} f_{ij} \cdot d_{\pi(i)\pi(j)} \tag{3.10}$$

The constraints of the assignment are respected by definition because of the use of permutations.

Complexity of the QAP

As already mentioned, the QAP has been proved NP-hard [42] by Sahni and Gonzalez [93]. They argued that even finding a polynomial f-approximation algorithm for the problem is not possible.

Some authors have also studied the practical complexity from different points of view. Roucairol *et al.* in [74] studied the difficulties encountered by exact methods when solving the QAP. They argued that the difficulty of this problem for exact methods of type branch-and-bound is also due to the structure of the instances. Mainly, the product of the two matrices may lead to many good quality solutions having very close cost. In such structure, it is hard for branch-and-bound algorithms to eliminate large subspaces in the tree. Related to this practical complexity, is the notion of flow dominance used by many authors to explain the hardness of some instances. According to the definition given by Angel *et al.* in [7], flow dominance is when few facilities exchange a lot of materials between them and few with the rest of the facilities.

From the metaheuristic's point of view, many authors pointed out that the variable results achieved by metaheuristics and local search algorithms on QAP instances are due to the structure of the latters. Angel *et al.* [7] have analyzed the hardness of the problem when

being solved by metaheuristics. The authors introduced new complexity measures based on the flow dominance measure. According to the authors, such complexity measures are intended for the use in the experimental stage of search algorithms being developed or to help on choosing the most adapted search method for each instance according to its complexity class. Another source of complexity for metaheuristics and local search algorithms is due to the combinatorial explosion of the search space when the size of the problem to be solved increases. For a size n problem, the number of feasible solutions is equal to $n!$. Thus, search algorithms need to be enhanced with diversity mechanisms in order to cover the whole search space and the use of large and multiple neighborhood structures for a good exploitation of promising subspaces.

3.3.1.2 Three-index assignment problem (AP3)

Origin and applications of the problem

The three index assignment problem (AP3) has been introduced in 1966 by Pierskalla [83]. In the first version of the AP3 as defined in [83], the problem consists in assigning a set of p items to q locations at r points or intervals of time in such a way to minimize the cost of the assignment and it is assumed that $p \leq q \leq r$. Among the possible applications reported in [83], one application consists in the scheduling of new investment's projects into different locations over a given period of time in such a way to minimize the total cost of the construction schedule. The definition of the AP3 used in recent works assumes that the three sets of the assignment have the same size: $p = q = r$. Furthermore, different versions and special cases of the problem have been studied in the literature, such as the axial 3-index assignment problem [16] and the planar 3-index assignment problem [72]. A maximization version exists also for the same problem [37].

Mathematical formulations of AP3

The 0-1 integer formulation of AP3 is stated as follows, given a $n \times n \times n$ cost matrix $C = (c_{ijk})$ where the elements c_{ijk} represent the cost of the assignment of the item i to the location j to the slot time k, the problem consists on minimizing the total cost of the assignment (Equation (3.11)) :

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{ijk} x_{ijk} \tag{3.11}$$

subject to

$$\begin{aligned} &\sum_{j=1}^{n} \sum_{k=1}^{n} x_{ijk} = 1,\ i = 1, \ldots, n \\ &\sum_{i=1}^{n} \sum_{k=1}^{n} x_{ijk} = 1,\ j = 1, \ldots, n \\ &\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ijk} = 1,\ k = 1, \ldots, n \\ &x_{ijk} \in \{0, 1\}\ \forall i, j, k. \end{aligned} \tag{3.12}$$

The decision variable x_{ijk} is equal to 1 if the item i is assigned to the location j in the interval of time k, 0 otherwise. The constraints of the assignment are given in Equations

(3.12). Each item is assigned only to one location at a unique slot time.

If we consider the assignment of adjacent items to locations at the different time periods, a permutation-based formulation of the AP3 could be stated as explained in Equation (3.13).

$$min \sum_{i=1}^{n} c_{i\pi(i)\psi(i)} \tag{3.13}$$

With π and ψ two permutations representing a feasible solution for the problem. Every single assignment is represented by a triplet $(i, \pi(i), \psi(i))$. The constraints of the assignment are all respected by definition because of the use of permutations. A solution for this problem is represented using a pair of permutations, thus the size of the search space is $n!^2$ for a problem of size n.

Complexity of AP3

The complexity of AP3 has been studied by Frieze *et al.* in [36] and this problem has been proved NP-complete since it is a special case of the set partitioning problem which is NP-complete [42].

3.3.1.3 Three dimensional quadratic assignment problem (Q3AP)

Origin and application of Q3AP

According to Hahn [47], the Q3AP has been introduced in the first time by William Pierskalla in 1967 in a technical memorandum, but has never been published in the open literature. The problem was re-discovered in 2004 by Hahn *et al.* [95, 47] while working on a data transmission system design. The problem consists in finding bit-to-symbol mappings for two (re)transmissions in the Hybrid Automatic Repeat Request protocol (HARQ) that minimizes the BER (Bit Error Rate) of the two transmissions. According to the explanation given in [109] and [95], in data transmission systems, when a data packet is sent through variable fading channels, some errors may remain in some bits of the received message, even after error correction mechanisms are used. The HARQ mechanism consists on sending to the transmitter a retransmission request for the same packet. Knowing that the reliability of the different transmission channels to use is variable, it is profitable to use a different mapping between bits of the message to send and transmission channels in order to create a diversity in the mapping and increase the throughput of the transmission. For example, for a system using QAP constellations (quadratic amplitude modulation), for each transmission, a different mapping should be used between the n modulation symbols and n segments of the message to send. Let us consider that the message to send contains 4 bits, the first bit and third bit transmitted through reliable channels are correctly decoded at the receiver, while errors remain for the second bit and fourth bit. In the second transmission, a different mapping between bits and channels is used which increases probability that the second bit and fourth bit will be received correctly. The correct message could be reconstructed at the receiver from the two transmissions. More information about the diversity mapping in the HARQ can be found in [95] and [109].

Mathematical formulation of Q3AP

Q3AP could be seen as a generalization of QAP to a third dimension. If we use the facility location example, a Q3AP problem could be seen as a facility to location to managers assignment. Indeed, as illustrated in Equation (3.14), Q3AP is formulated in [47] as an extension of the formulation of QAP. If we consider a diversity mapping problem in which a mapping should be used between N modulation symbols and N segments of the message to send, the cost coefficients C_{ijpknq} in Equation (3.14) refer to the simultaneous assignment of the symbols j and n to the bit strings i and k in the first re-transmission and the assignment of symbols p and q in the second re-transmission. Equations (3.16), (3.17) and (3.18) express the uniqueness constraints of the two assignments (one symbol to one segment and *vice versa* in the two bit-to-symbol mappings).

$$min \left\{ \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{N} b_{ijp}x_{ijp} \; + \; \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{N}\sum_{k=1}^{N}\sum_{n=1}^{N}\sum_{q=1}^{N} C_{ijpknq} \; x_{ijp} \; x_{knq} \right\} \tag{3.14}$$

$$x \in I \cap J \cap P, \; x = 0, 1 \tag{3.15}$$

With: I, J, P sets of same cardinality N, and:

$$I = \left\{ \; x \geq 0 : \sum_{j=1}^{N}\sum_{p=1}^{N} x_{ijp} = 1 \quad for\, i = 1, .., N \; \right\} \tag{3.16}$$

$$J = \left\{ \; x \geq 0 : \sum_{i=1}^{N}\sum_{p=1}^{N} x_{ijp} = 1, \quad for\, j = 1, .., N \; \right\} \tag{3.17}$$

$$P = \left\{ \; x \geq 0 : \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ijp} = 1, \quad for\, p = 1, .., N \; \right\}. \tag{3.18}$$

According to Hahn *et al.* [47], artificial Q3AP instances could be obtained from QAP ones using Equation (3.19). F and D are the two matrices representing the flow and the distances between factories/locations in QAP instances.

$$C_{ijpknq} = F_{ik} * D_{jn} * F_{ik} * D_{pq}; (i, j, p, k, n, q = 1..N) \tag{3.19}$$

If we consider the assignment of N message segments to N adjacent modulation symbols during two transmissions, a solution for Q3AP can be represented by two permutations π and ψ of numbers in the set $\{1, 2, \ldots, N\}$ referring to the labels of the N message segments to send during the two transmissions. The permutation formulation of Q3AP is derived from the permutation formulation of QAP (see Equation (3.20)). $\pi(i)$ and $\psi(j)$ respectively, indicate the bit string assigned to the modulation symbol i and j respectively in the first and the second transmissions respectively.

$$minf(\pi, \psi) = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{i\pi(i)\psi(j)j\pi(i)\psi(j)} \tag{3.20}$$

Complexity of Q3AP

The Q3AP formulation is an extension of the QAP and the 3AP ones, which are both NP-hard [93, 36]. Consequently, Q3AP is necessarily NP-hard. Furthermore, for Q3AP instances which are derived from QAP, the flow dominance and other related complexity measures defined for QAP, could also be applied to Q3AP. On the other hand, in the permutation representation of Q3AP, the combinatorial size of the search space of this problem is estimated to $n!^2$ for a problem instance of size n.

3.3.2 Scheduling problems

Scheduling problems are widely used in some industrial domains for the modeling of industrial production processes. In this section, we discuss some scheduling problems which can be represented using permutations: the permutation flowshop and the jobshop problems.

3.3.2.1 Permutation flowshop problem (FSP)

Definition and formulation of the permutation flowshop problem

The permutation flowshop problem is one of the numerous scheduling problems. According to the definition given by Brucker in [13], the general flowshop problem consists in the scheduling of a set of n jobs $J_1, J_2, \ldots, J_n$ on m machines. Each job i consists of m consecutive operations (or tasks) O_{ij} with processing times p_{ij} $(j = 1, ..., m)$ where O_{ij} means the j^{th} operation of the job J_i must be processed on machine M_j. There is a constraint of precedence between the operations ($O_{ij} \rightarrow O_{i,j+1}$) and each machine is allocated to a unique job at the same time. The problem consists in finding a sequence order of jobs π_j, for each machine j, $(j = 1, \ldots, m)$ in such a way to minimize the total completion time.

In this thesis we are concerned only with a special case of the flowshop problem, the permutation flowshop. The latter is considered when the jobs are scheduled in the same order on all the machines, which means

$$\pi_1 = \pi_2 = \cdots = \pi_m$$

A processing time p_{ij} and a starting time s_{ij} is associated to each operation O_{ij}. The permutation flowshop consists in finding a permutation $L =< L_1 L_2 \ldots L_n >$ which represents the order in which the jobs will be executed on the m machines. The objective is to minimize the *makespan* (C_{max}) which represents the total completion time of the n jobs on all the machines (see Equation (3.21)).

$$C_{max} = Max\{s_{iM} + p_{iM} | i \in [1 \ldots n]\} \tag{3.21}$$

Complexity of the problem

If the number of machines $m = 2$ the problem can be solved in a polynomial time $O(n\ log\ n)$ using the Johnson's algorithm [53]. For $m \geq 3$, the problem is NP-complete

[42] and finding optimal schedules requires an enumeration method such as B&B to implicitly enumerate the $n!$ possible permutations.

3.3.2.2 Jobshop problem (JSP)

The jobshop problem is a generalization of the permutation flowshop problem. We refer again to the definition given by Brucker [13]. We are given n jobs $J_i, J_2, \ldots, J_n$ to schedule on m machines $M_1, M_2, \ldots M_m$. Each job J_i is composed of n_i operations referred to as O_{ij}, $i = 1, 2, \ldots, n$ and $j = 1, \ldots, n_i$. The sequence of operations $O_{i1}, O_{i2}, \ldots, O_{in_i}$ of each job must be processed in this order on all the machines (precedence constraint). Let p_{ij} be the processing time of the operation O_{ij} on the machine M_{ij} with $j = 1, \ldots, n_i$. It is assumed that each operation O_{ij} is processed on a different machine M_{ij}, which means $M_{ij} \neq M_{i,j+1}$ for $j = 1, \ldots, n_{i-1}$. The job shop problem consists in finding a feasible schedule which minimizes the completion time or the makespan C_{max} of the last operation scheduled.

A permutation representation for JSP is possible using multi-permutations [10, 85]. Each permutation represents the sequencing of operations on one machine. The total number of operations is equal to $n \times m$ with n the number of jobs and m number of machines. Nevertheless, it has been pointed out by Bierwirth in [10] that due to the different constraints, it is impossible to represent JSP using standard permutations without including infeasible solution in the coding. For example, if we consider a representation using a multi-permutation as described earlier, it may easily happen that two jobs scheduled in two different machines break the precedence constraint. To circumvent this obstacle, some authors use weight penalties to progressively discard infeasible solutions from the search, or use reparation to transform an infeasible solution into a feasible one. Nevertheless, should this happen too often, the search will be disrupted. In [10], a new representation for JSP is presented. This representation uses a single permutation with repetition to represent the sequencing of all the operations on all the machines.

With this representation each solution needs a phase of decoding (or reading) before it is evolved. In this representation, each job number appears m times in the permutation, which means, as often as there are operations belonging to this job. In the representation theory, this is known as an indirect representation. Classical neighborhood and genetic operators can not be used without a decoding step, that is the permutation should be read according to the properties of the problem.

3.3.3 Traveling salesman problem (TSP)

Like QAP, the traveling-salesman problem (TSP) is a well studied problem in the literature due to its hardness and the number of its applications [42]. In the TSP we are given a set of n cities and a cost matrix which contains the traveling cost between each pair of cities and we are asked to find the least cost tour of cities in which all the cities are visited exactly one time. Formally, the problem is generally stated as a graph $G = (V, A)$ where $V = \{v_1, \ldots, v_n\}$ is the set of vertex and $A = \{(v_i, v_j) : v_i, v_j \in V\}$ is the set of arcs. A cost c_{ij} (or a weight) is associated to each arch (v_i, v_j). The problem consists in finding a Hamiltonian circuit which minimizes the overall cost of the arcs involved in the circuit. There

are several variants of this problem. For example, the symmetric TSP (STSP) is considered when the cost of each vertex satisfies the following property

$$c_{ij} = c_{ji} \ \ \forall i, j \in \{1, 2, \ldots, n\}$$

The graph is then undirected as changing the direction between two cities in a tour does not change the cost of the arc. Conversely, the asymmetric TSP (ATSP) is viewed as a complete directed graph as each direction in the same arc has two different weights. The permutation representation of TSP is viewed as a sequence of cities placed in a permutation π of size n, according the order in which they are visited in the tour. And since it is a tour, the first city (in the first entry of the permutation) is connected to the last one. The permutation formulation of the problem is given in Equation (3.22). The overall cost of a feasible tour is the sum of the distances between each couple of adjacent cities in the tour.

$$Minimize \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)} \tag{3.22}$$

3.4 Towards classes of permutation-based problems and appropriate search operators' properties

Metaheuristics are search algorithms for general use but for more efficiency, the basic search operators should respect the constraints and properties of the tackled problem. For example, when solving QAP instances the insertion operator used for flowshop will not be efficient because after one single insertion move, a large section of the permutation is modified. The resulting permutation is different compared to the initial one in many positions. However, the objective function of the QAP is based on the absolute positions of the elements. That is, the permutation is not interpreted in the same way in the different problems. In this section, we present a classification for the permutation-based problems presented in this chapter.

In [56], Kargupta *et al.* presented a study of GA operators for *deceptive ordering problems*. Ordering or sequencing problems are generally represented by permutations but not all genetic operators applied to permutations are efficient for sequencing problems. Three different crossovers were studied and the schema survival probability for each operator is calculated. That is the probability that a block of genes of high quality, by respect to the problem being solved, will survive and be transmitted to the offspring. The considered operators are classified as *absolute-position*, *relative-order*, and *absolute order* operators following the fact that they transmit only absolute positions of the genes, relative order between two genes or both of them.

In this chapter, we reuse these three categories in order to classify the permutation-based problems presented in Section 3.3. This classification is shown in Figure 3.1. In order to classify the permutation problems, we used their respective definitions and objective functions. The objective function allows us to know which kind of information encoded in

the permutation, is meaningful for the problem at hand. The tree classes are further defined in the sequel of this section.

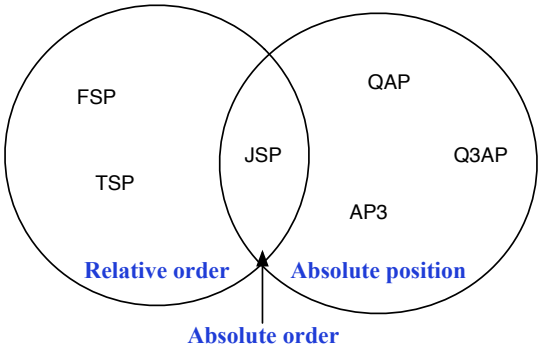


Figure 3.1: Classification of some permutation problems according to the significance of the absolute positions and relative order of items in the permutation.

3.4.1 Absolute position problems

This class regroups all assignment problems (QAP, Q3AP, AP3, etc.). As already mentioned, the permutation representation is interpreted differently in each problem. In assignment problems, it represents the arrangement without repetition of a set of n objects. Each elementary assignment is exclusively defined by an object i and its associated position j in the permutation. The absolute position assigned to each element in the permutation is associated to a cost which influences on the overall cost of the solution. For quadratic assignment problems, the pair-wise assignment of two pairs of elements is also significant because the quadratic cost of each assignment is defined in taking into account the rest of assignments. Thus, this property should be taken into account when designing search operators for assignment problems.

3.4.2 Relative ordering problems

In this class of problems, the permutation is interpreted as a sequence. The cost of a sequence is determined by the cost of the elementary subsequences which are composed of any two adjacent elements in the permutation. In the TSP, the last element is also related to the first one. In this class one can classify all sequencing problems such as TSP and the permutation flowshop. Indeed, in the flowshop, the permutation is an encoding for a sequence of jobs to be scheduled in this order. The permutation representation is used to guarantee the feasibility of all the schedules: each job is executed in the M machines only once.

3.4.3 Absolute ordering problems

Unlike the two previous classes which represent two extreme cases, there are a variety of problems which are in between the two classes. The problems in the class absolute ordering

problems relay both on the positions and on the order of the components in the permutation. For example, in [10] Bierwirth argued that in the case of JSP, the order between two elements is significant because it represents the order in which the jobs are scheduled in a given machine. On the other hand, the absolute position of the jobs is important because it expresses the precedence relations among operations in the schedule. Thus, both absolute position and order are important.

3.5 Neighborhood and genetic operators for permutation-based problems

After presenting the different classes of permutation-based problems and their properties, in this section examples of search operators including neighborhood and genetic operators applied to each class of permutation-based problems, are presented.

3.5.1 Neighborhood operators

Most metaheuristics used for permutation-based problems are based on neighborhood search implemented by inner local search procedures guided by the high level metaheuristic. Some work references are studied here for different problems. In [101], Stutzle reported an ILS algorithm for QAP in which the local search step is based on the so-called 2-exchange neighborhood. This neighborhood is also used for the other assignment problems: Q3AP [47], [104], AP3 [81],[2].

In [92], Ruiz *et al.* presented an iterated greedy algorithm for permutation flowshop problem in which an insertion neighborhood is used. Ben-daya and Al-Fawzan [9] proposed a tabu search approach for the same problem and studied the performances of three different neighborhoods: swapping of two jobs, insertion of one job in a given position and block insertion in which a sequence of jobs is inserted in a given position. The three neighborhoods are combined. In [100], Stutzle also studied three neighborhoods for the flowshop: the swap of two jobs $\pi(i)$ and $\pi(i+1)$, the exchange of two jobs $\pi(i)$ and $\pi(j)$ and the insertion neighborhood. According to Stutzle, the swap is the fastest neighborhood but it yields poor quality schedules. The pair-wise exchange neighborhood and insertion yield equal quality solutions but the insertion is evaluated more efficiently for the permutation flowshop. In [100], the insertion neighborhood is used in the local search procedure of an ILS while the swap or pair-wise exchange are used in the perturbation step in order to avoid reversing the insertion move.

For the TSP, many metaheuristics for this problem are based on the Lin-Kernighan Heuristic [68] which consists of the swapping pairs of sub-tours to make a new tour. Simpler neighborhoods uses the graph representation of the TSP in order to design effective moves on the permutation representation such as the 2-opt and k-opt neighborhoods. The 2-opt operator is equivalent to the 2-exchange operator used for assignment problems. The k-opt neighborhood is a generalization for the 2-opt. These two operators are also very often used for other permutation problems. For example, the 3-opt is used as a perturbation operator in ILS

algorithms applied to QAP and Q3AP [101] ,[47]. In the rest of this section we will present two neighborhoods for permutation problems: 2-exchange or 2-opt and insertion.

3.5.1.1 Pair-wise exchange/2-opt

In the TSP, the neighborhood $N(s)$ of a valid tour s is obtained by replacing two edges (i, j) and (j, l) in the initial tour by two edges (i, l) and (l, j) (see example in Figure 3.2).

In terms of permutations, this is equivalent to the pair-wise exchange of two elements in the positions l and j. In the example illustrated in Figure 3.2, the exchanged edges are: (c_5, c_4) and (c_4, c_2) by the edges (c_5, c_2) and (c_2, c_4). The orientation of the edge (c_4, c_2) is also changed. This last modification is the one which leads to a swapping between two adjacent positions in the permutation representation of the problem. However, the only significant change for the TSP is the adjacency of the elements 5 and 2 representing the edge between the cities c_5 and c_2 in the new tour. The absolute positions of the cities in the tour is meaningless.

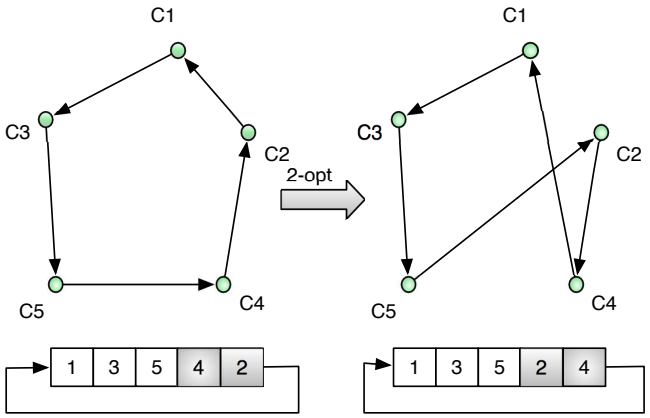


Figure 3.2: Illustration of the 2-opt neighborhood on a TSP example

The pair-wise exchange neighborhood can be built either by randomly selecting two elements in the permutation to swap or by a systematic enumeration which consists of exchanging the positions of each couple of elements in the permutation. The size of this neighborhood is $n(n-1)/2$. The 3-opt neighborhood is obtained by exchanging the positions of 3 elements in the permutation. It is equivalent to executing the 2-opt twice. If π is the initial permutation, a neighbor of π is obtained by exchanging $\pi(i)$ and $\pi(j)$ followed by the exchange of the elements $\pi(j)$ and $\pi(k)$. The k-opt is a generalization of the 2-opt to k elementary moves in the permutation.

3.5.1.2 Insertion

The insertion neighborhood is typically a relative order neighborhood as it relays only on the relative order between the elements in the permutation. The permutation in this case is viewed as a sequence. The insertion neighborhood is defined as follows. Given a permutation π of size n, neighboring sequences for π are obtained by inserting the element

$\pi(j)$ in the position i in the permutation. By definition, the elements $\pi(k)$ with $k > i$ are shifted by one position to the right (if $j > i$). The position j of the element to insert and its new position in the permutation (i) can be determined either randomly or by a systematic way. For example, to build the entire insertion neighborhood of the permutation π one may start by inserting every elements $\pi(j)$, $j = 1, \ldots, n$ in every position $i = 1, \ldots, n$ with $i \neq j$. The size of this neighborhood is $(n-1)^2$.

An illustration of this neighborhood is shown in Figure 3.3. This neighborhood is not efficient for absolute position problems because the neighbors are different from the initial solution s in many positions.

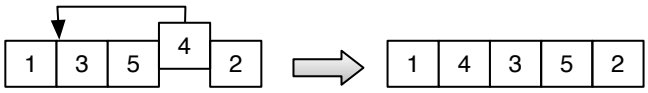


Figure 3.3: Illustration of the insertion neighborhood.

3.5.2 Genetic operators

In this section we try to review at least one crossover operator and one mutation for each class of permutation-based problems. For this, we studied a set of works for each class. For absolute position problems, the most used crossover operators are the partially mapped crossover (PMX) and the position crossover (PX). For relative order problems, in [79] Oliver *et al.* presented a study of crossover operators used on the permutation representation of the TSP. In the same subject, Kuijpers *et al.* in [64] reported a review of different genetic operators (crossovers and mutations) for the TSP on different representations (binary, permutations). The two works converge to a relative order crossover operator in which only the order of cities on a tour is important. In [56], Kargupta *et al.* presented a relative order crossover (ROX) in which only the relative order between the elements of the permutation is important.

For the class absolute order problems, Bierwirth [10] studied two crossovers for JSP: a Generalized Order Crossover (GOX) and a new crossover called Precedence Preservative Crossover (PPX). The two crossovers work on a permutation representation with repetition. The objective of PPX is to take into account the precedence relations between the jobs when scheduled in two different machines. In [56], Kargupta *et al.* also studied an absolute order crossover, the Uniform Order Crossover (UOX). For mutations, generally the swap mutation is very used for any kind of permutation problems. The other mutation studied in this section is the shift mutation also used for all classes of permutation problems. In the following, we will present a selection of operators including the crossovers PMX, ROX, UOX and the mutations swap and shift.

3.5.2.1 Partially Mapped Crossover (PMX)

This crossover is initially proposed by Goldberg and R. Lingle in [45]. PMX tries to respect the absolute positions of the two parents when creating the offspring. The two parents P_1 and P_2 are used alternatively as first and alternate parents to create the two childes C_1 and C_2. Two cutting points are selected uniformly in the two parents and the crossover proceeds as illustrated in Figure 3.4.

In the first step, the positions between the two cutting points in the offspring are filled from the first parent. Second step, we try to fill the remaining positions with the corresponding elements in the same positions but from the alternate parent, as long as there is no feasibility constraints violated (duplicated elements). In the example, the elements 6, 2 and 5 are placed in the same absolute positions as in the alternate parent. The elements 4 and 7 from the alternate parent are duplicate, so those positions are filled using mapping relations of the alternate parent to the first parent. The elements 4 and 7 in the first parent correspond respectively to the elements 8 and 3 in the alternate parent. The element 3 is placed while 8 is duplicated. Finally, the first position in the offspring is filled from the remaining element, 1.

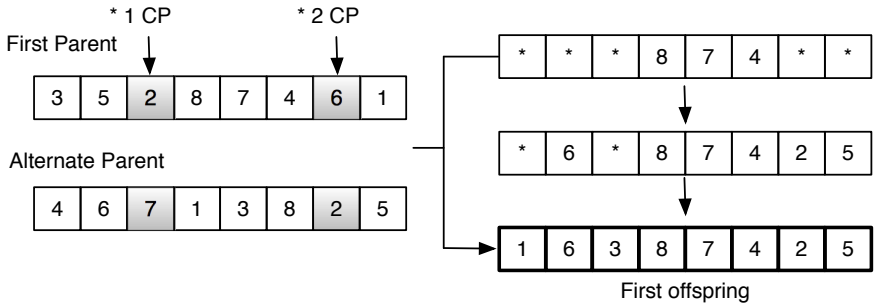


Figure 3.4: PMX crossover example

3.5.2.2 Relative Order Crossover (ROX)

This crossover is reported by Kargupta *et al.* in [56]. It is designed by the authors specially to evaluate the performances of an operator which respects exclusively the relative order of elements in an ordering deceptive permutation problem. This crossover is implemented as follows:

- A set of k alleles is randomly selected in the first parent (the elements 5, 2, 6 in the example Figure 3.5). This is called an *order matching section*.
- A preserve list P is used to store the elements of the order matching section by preserving their order and a waiting list W is used to temporarily store some elements from P.
- The second parent is scanned position by position from the left. If the element at the current position is not in P, it is copied to the child (from left to right), if it is a member of P but not the first in the ordering, it is appended to the set W, otherwise,

it is copied to the child and deleted from P. If the next element in P is also a member of W, it is copied to the child. This operation is repeated until the child is completely filled.

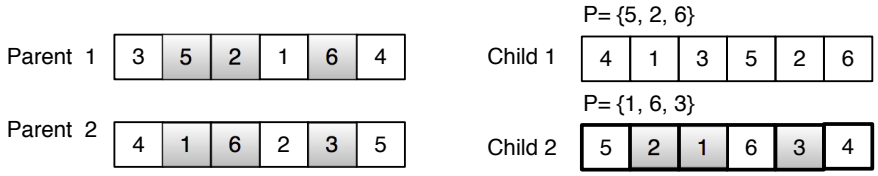


Figure 3.5: ROX crossover example.

3.5.2.3 Uniform Order Crossover (UOX)

This crossover was initially proposed by Davis [23] and is referred to as Order Crossover (OX) in the literature. The same crossover is called Uniform Ordering Crossover (UOX) in [56] and the name OX is used for another crossover for permutations, which unlike the OX of Davis, does not preserve absolute positions of elements. To differentiate between the two crossovers, we will use the notation of Kargupta *et al.* [56] UOX. UOX was initially designed for the TSP. It conserves a sub-tour from the first parent (absolute positions) and the relative order of the remaining elements from the other parent. A mask of bits of the same length than the parents is randomly generated. All the elements in the first parent which correspond to the bit 1 in the mask, are copied to the offspring. The remaining unplaced elements from the first parent (those corresponding to the 0 in the mask) are ordered as they appear in the alternate parent and placed in this order in the remaining positions of the offspring (see Figure 3.6). The second child is created using the same procedure after permuting the two parents.

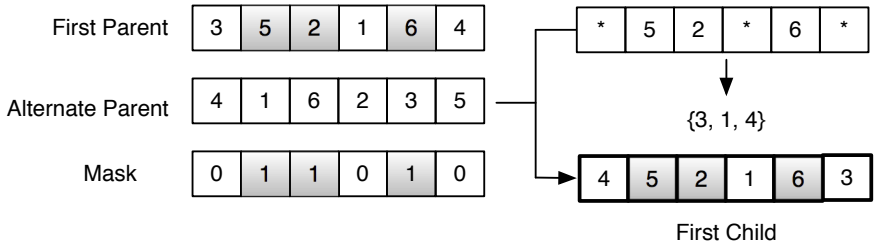


Figure 3.6: UOX crossover example.

3.5.2.4 Shift mutation

This mutation operator is inspired from the insertion neighborhood. Given a permutation π, the elements in the section determined by two randomly chosen positions i and j are shifted

by one position to the left or to the right. This is equivalent to the insertion of $\pi(i)$ in the position j.

3.5.2.5 2-exchange mutation

Same remark than the shift mutation, the 2-exchange mutation is equivalent to the 2-exchange neighborhood. Two random positions i and j are selected in the permutation and the elements $\pi(i)$ and $\pi(j)$ are permuted.

3.6 Conclusion

In this chapter, permutations as mathematical objects are studied and some important properties for metaheuristics are outlined. These properties can be used in the design of different tools in metaheuristics such as distance measures and search operators. A non exhaustive list of permutation-based problems including assignment problems, scheduling and TSP are studied and classified into three classes according to the interpretation made of the permutation representation. The absolute position class contains all assignment problems. The absolute position of each element in the permutation is meaningful as it participates in the definition of the overall cost function of a complete assignment. For this class, the adequate neighborhood and genetic operators are those which respect this property and transmit the absolute positions of genes to the offspring. More exactly, the 2-opt or 2-exchange and the swap neighborhoods as well as the PMX crossover are the most used operators for absolute position problems. The second class of permutation problems is the relative order class. TSP and sequencing scheduling problems such as flowshop, are classified in this category. The permutation is viewed as a sequence of jobs in a schedule or a set of cities in a valid tour of the TSP. The adequate search operators for this category allows to respect the relative order of genes in the sequence. Among relative ordering operators, in this chapter we studied the insertion neighborhood and the ROX crossover. Finally, between these two extreme classes of permutation-based problems, there are some problems for which both relative order and absolute positions of genes is meaningful. This class of problems called absolute order problems, contains constrained scheduling problems such as JSP. The search operators applied for this class in the literature are respectful of both relative order and absolute position of genes, such as in UOX crossover.

Most of the permutation-based problems studied in this chapter are challenging for both exact and approximate search methods. In particular, quadratic assignment problems are among the hardest optimization problems. For example, the size of the largest QAP benchmark solved to optimality is around 32 and for Q3AP, the largest optimally solved problem is of size 16. The complexity of permutation-based problems is partially due to the combinatorial explosion which occurs when the size of the problems increases. Thus, traditional search algorithms are limited to small and average size problems. In this thesis, we study different hybridization strategies for permutation-based problems. Particularly, hybrid methods combining exact search methods with metaheuristics are promising as the two classes of search algorithms offer divergent and complementary properties.

Part II

Thesis Contributions

Chapter 4

Tree-based exploration of the search space in metaheuristics

Contents

4.1 Introduction

In this thesis, we are interested in the use of the properties of permutations such as Lehmer codes and inversions, and factorial system to convert an optimization problem represented by permutations into a problem represented by numbers. Indeed, as already quoted in Section 3.2.5, Lehmer code and factorial number systems can be used to convert a permutation of size n to a unique integer number in the interval $[0, n![$. This number represents the sequence order of the corresponding permutation in the process of enumeration of all the permutations in a lexicographic order. Because the mapping function from permutations to numbers and *vice versa* is a bijection between the set S_n of all permutations of size n and the set of integer numbers in the interval $[0, n![$, one may use numbers to represent the candidate solutions in permutation-based problems instead of the representation based on permutations.

The main motivation for this idea is that this type of representations might help search methods for permutation-based problems to work in a simplified search space instead of the complex search space of permutations, especially for problems using more than one permutation to represent candidate solutions (generalized permutation problems). This representation could be used for different purposes in different classes of optimization methods (exact methods and metaheuristics). It can also be of a great utility in the context of hybrid methods that combine exact methods with metaheuristics. Mainly, this special representation could be used as a connection between the divergent search strategies in the two methods in order to define a common global guidance of the search in the hybrid algorithm and to take advantage from each method.

However, according to the theory of representation [91], the representation used to model a given optimization problem is very important for the efficiency of the search algorithm, because it may influence the efficiency and effectiveness of the problem solving in the good or the bad way. In this context, the most important feature in the theory of representation is the concept of locality. According to Rothlauf [91], in the context of evolutionary algorithms, the locality of a representation describes how well the phenotype metric corresponds to the genotype metric. A representation with high locality is a representation in which neighboring genotypes correspond to neighboring phenotypes. Conversely, a representation with low locality is a representation in which the neighboring genotypes are not necessarily neighbors in the phenotype space and the use of this type of representations in a search method may lead to a random search.

In this chapter, first we study the new representation for permutation-based problems both in the context of exact methods and metaheuristics. Possible use cases of this new representation in the two classes of optimization methods are explored in the light of related works from the literature. Then, as we are interested in permutation-based problems with any number of permutations, we propose a generalization for the mapping algorithms from permutations to numbers and *vice versa*. Finally, search operators for metaheuristics based on the new representation are proposed and designed in the light of the theory of representation.

4.2 In the context of exact methods: a coding approach for permutation trees in the branch-and-bound algorithm

In order to enumerate all the solutions of the solution space, the B&B algorithm progressively builds a tree that covers the search space. The root node is the initial problem to be solved of size n and intermediate nodes are sub-problems of size $n - d$ (d is the depth in the tree). Finally, the leaves represent complete solutions of the problem. In [76], a special coding for this tree is used to facilitate the decomposition of the search space between several B&B processes. Indeed, each node in the tree is assigned a unique number (identifier) and any set of contiguous nodes (having successive numbers) could be represented as an interval. This scheme transforms the complex search space of permutation problems to a one dimensional search space.

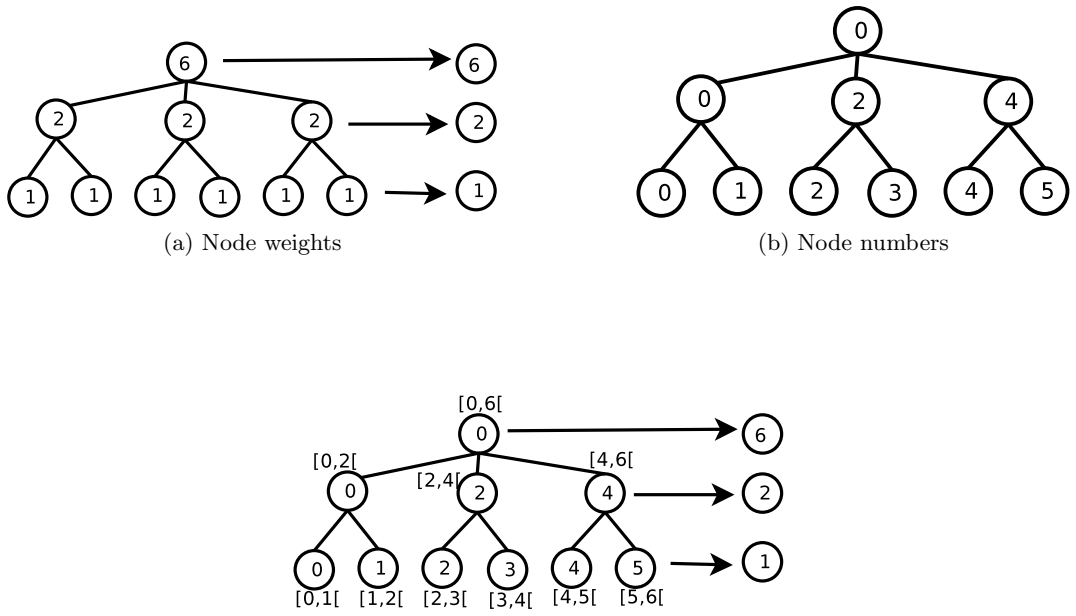
The node's path, node's weight and node's rank define the number associated to any node in the tree. The number associated to any node in a permutation tree is calculated using Equation (4.1) where $path(p)$ is the set of nodes from the root to the node p, including both the root and p, $weight(p)$ is the number of leaves of the sub-tree of the node p computed by Equation (4.2), and $rank(p)$ is the position of a node p among its sibling nodes. Using this approach, any set of contiguous nodes can be represented by an interval as shown in Figure 4.1 (c) which is an example for coding a one-permutation tree of size $n = 3$. Assuming that n is the size of the permutations, the size of the search space is $S = n!$ and the whole search space can be represented by the global interval $I = [0, n![$. In Figure 4.1 (c), the global interval is equal to $[0, 6[$. Finally, the concept of range of p, noted as $range(p)$, defines the interval that contains all the nodes for which the node p is the root node (see Equation (4.3)). This method is used to share the global search space interval between several B&B processes. Each process will get a sub-interval to explore. A special operator named *unfold* is used to transform any interval to a minimal set of nodes in each B&B process. This coding approach facilitates the parallelization of the B&B algorithm and its deployment on a distributed system such as computational grids.

As pointed out in [76], the use of the natural numbers and especially the concept of interval to represent a set of tree nodes instead of listing all the nodes to be visited, helps to simplify the mechanisms related to the parallelization of the B&B and its deployment on a large scale system such as: communication optimization (only intervals are sent through the network instead of hundreds of tree nodes), load balancing between the available computing resources (a work unit is represented by an interval and is shared between the parallel B&B processes) and the detection of the termination (the search is ended when there is no more intervals to be explored).

$$number(n) = \sum_{i \in path(n)} rank(i) * weight(i) \tag{4.1}$$

$$weight(n) = (n - depth(n))! \tag{4.2}$$

$$range(n) = [number(n), number(n) + weight(n)[\tag{4.3}$$



(a) Node weights

(b) Node numbers

(c) Intervals

Figure 4.1: The tree-based representation where each node has a unique number and contiguous nodes are represented by intervals.

In this thesis, we focus on all permutation-based problems regardless of the number of permutations used. Thus, firstly we need to generalize this B&B coding to permutation-based problems with m permutations, $m > 0$. For example, we consider the Q3AP in which a solution is represented by two permutations. Because the structure of the tree does not change if we consider more than one permutation, the equation used to calculate the numbers associated to each tree node will remain the same, see Equation (4.1). However, if m permutations are used for the tackled problem, the size of the search space is $n!^m$, n being the size of the problem. Thus, the weight of a node p at a depth $depth(p)$ in a permutation tree with m permutations is given in Equation (4.4). The global interval which represents the whole search space of the problem in this case is: $[0, n!^m[$.

$$weight(p) = (n - depth(p))!^m \tag{4.4}$$

If we look slightly at Equation (4.1) we notice that it is equivalent to the equation used in factorial number systems to convert a factorial number into its decimal representation (see Equation (3.6) in Section 3.2.5). For example, the weight of a node at the depth i in the permutation tree is nothing but the radix of the i_{th} digit from the right in the factorial number. Mainly, the authors in [76] have used exactly the same technique to generate the numbers associated with the nodes in the tree but only using the properties of permutation trees, without any reference to factorial numbers or to Lehmer codes. In the rest of this thesis, we will use the two references indifferently: coded tree representation and numbering of permutations. In some cases, the tree is useful to understand some properties of permutations. In the following, we discuss the use of numbers to represent permutations in the context of

metaheuristics.

4.3 Motivations for the numbering of permutations in metaheuristics

In metaheuristics, this type of coding for permutations is not common. Indeed, very few works deal with the subject in the optimization community.

In [82], Lehmer code is used to represent permutations in a differential evolutionary algorithm to solve a traveling salesman problem with constraints. The individuals are represented by both permutations and their associated Lehmer codes. The permutation representation is used to compute the fitness of the individual and the Lehmer code is used in the differential evolution. Indeed, $\pm$ operations on permutations via their Lehmer code are defined by the authors to be used in the differential evolution. In this work, the Lehmer code plays a major role in the algorithm as it is used in the differential evolution.

In [94], a Particle Swarm Optimization algorithm (PSO) for single row facility layout problem is proposed. A mapping between the discrete space of permutations and the continuous space of natural numbers via the Lehmer code, is used to represent candidate solutions for the problem. The PSO uses this coding to explore the continuous space. Mainly, the solutions are represented by natural numbers in the interval $[0, n! - 1]$ during the search. Afterwards, a decoding algorithm is used to go back to the permutation's discrete space in order to calculate fitness function and/or to return the final solution.

In the context of metaheuristics, two questions should be answered: (1) What is the advantage took from the use of Lehmer code and numbers to represent permutations in metaheuristics? (2) What is the cost of coding/decoding operations required each time we need to calculate the objective function for every candidate solution?

For the first question, in both works, the use of numbers and Lehmer codes during the search instead of permutations, facilitates the design of the required operators: $\pm$ operations on permutations via their Lehmer codes in the differential evolutionary algorithm in [82] and the process of updating the velocity in the PSO in [94].

Our own motivations for using this representation in the context of metaheuristics can be summarized in two points. First, the restriction of the search to a given sub-space represented as an interval can be useful in many cases such as in the case the search space to be explored is too large for the available time. Therefore, it can be more interesting to focus the search on a part of the search space and intensify it to find good local solutions. Another interesting application is the parallelization of the metaheuristic using the same parallelization techniques used in tree-based exact algorithms: the search space is explicitly shared between the parallel searches. Which means, instead of having a set of heuristic search agents which are executed in parallel and are supposed to search conjointly in the global search space in the same time, one could share this search space between the search agents and each heuristic will be focused only on its own sub-space. One could also have a constellation of search agents on each sub-space as the cooperation between the parallel

search agents on a single sub-space could help on diversifying their search. For example, in the case of the parallel GA model known as island model, each island will be focused only on a single sub-space of the global search space of the tackled problem. No communication is necessary with the islands working on the other sub-spaces because every island is working on a different problem. Each sub-space will be (hopefully) searched more efficiently by the associated island (or set of cooperating islands).

The second motivation for this type of representation is that since the same representation is used in some tree-based exact search methods such as B&B, this may help on the design of efficient and powerful hybrid search methods combining at low-level tree-based exact search algorithms and metaheuristics. The two methods could exchange very accurate information because each method is aware about the search of the other method and about the historic of the global search. This is also a good way to diversify the search of the metaheuristic to unexplored sub-spaces.

Regarding the second question (what is the cost of coding/decoding operations required each time we need to calculate the objective function for every candidate solution), as already noticed in Section 3.2.5, transforming a permutation to Lehmer code then to integer can be done in a polynomial time.

Nevertheless, a third question related to the nature of the search space in metaheuristics arises of itself: Is the neighborhood explored by search operators in the continuous space homogeneous with the neighborhood in the permutation space? In other words, two neighboring solutions in the continuous space are they neighbors in the permutation space as well? In the theory of representation, this property is known as locality of a representation [91].

Mainly, it is known that the principle of move operators in metaheuristics is to perform a small change in the actual solution to get a new solution which is at a minimal distance from the fist one. The reason is that good quality solutions are generally grouped in the same area. Thus, they must share the same good quality components. In our concern, if the operator used in the continuous space (where solutions are seen as numbers) produces a solution which is completely different than the current solution in the permutation space which means the distance between the two solutions is big, the search will be equivalent to a random search!

In the rest of the chapter, we will study the locality of this new representation for permutations in the context of metaheuristics.

4.4 Adaptation of the coded tree-based search space representation to metaheuristics

In metaheuristics, the search space is explored in a different way compared to B&B. *Move* operators are used to go from one solution to another. The set of visited solutions using a given operator is called the *neighborhood* of the starting solution and there is no ordering notion between the neighbors. This is why the use of the tree-based representation in this context is quite different compared to exact methods such as B&B in which the algorithm consists in implicitly enumerating all the solutions of the search space. In order to take advantage from the coded tree-based representation and the numbering of permutations in

metaheuristics, we need to recall fundamental concepts in the theory of representation and to check if the two representations of the search space, the tree used in tree-based exact methods and neighborhood structures used in metaheuristics, could cohabit together in one global search method. In the following, key concepts in the theory of representation are recalled: the locality of a representation and the connectivity of neighborhoods. Afterwards, the tree-based representation of the search space and the use of a double representation in permutation-based problems in the context of metaheuristics, is studied according to the concepts of locality and connectivity.

4.4.1 On the locality of the tree-based representation

In this section, it is a question of checking if the lexicographic order used in the tree while enumerating the permutations, preserves the distances between the solutions in the permutation space. In other words, two adjacent solutions in the tree are they neighbors in the sens of the distance in the permutation space?

To answer this question, an illustration is given in Figure 4.2 for a permutation problem with one permutation of size $n = 5$. If the order used in the tree is lexicographic, then each level or depth d in the tree corresponds to the position d in the permutation. Let us consider the interval $I = [0, 30[$. Among the 30 solutions included in this interval, how many solutions are neighbors to the solution $s(03214)$ in Figure 4.2 according to the 2-opt operator and the Hamming distance? Two solutions s_1, s_2 are neighbors if $d(s_1, s_2) = d_{min}$. According to the Hamming distance on permutations, the minimal distance between two permutations is equal to 2 (minimal number of positions that are different in the two permutations). The distances between the target solution s and the set of solutions included in the intervalle I are calculated and neighboring solutions are highlighted in Figure 4.2. Among the 30 solutions of the interval, 6 solutions are at a minimal distance to the solution s. Notice that the size of the global search space in this case is $5! = 120$ and the size of the neighborhood engendered by the 2-opt operator is $n(n-1)/2 = 10$.

As a rule of thumb, if we consider a problem with one permutation of size n, two solutions s_1 and s_2, sharing the p first positions in the permutation (the roots until the depth $d = p+1$ in the tree) are separated by at most

$$(n-d)! \times (n-p)$$

solutions during the enumeration process. That is the weigh of a node in the depth d of the tree times the number of its sibling of the same depth. The number of sibling nodes in a given depth refers to the number of remaining variables to branch at this level of the tree. This formula supposes that the two solutions s_1 and s_2 are situated at the extreme borders of their respective roots.

In the case of general permutation problems where $m > 1$ permutations are used, the distance in the tree between the solutions s_1 and s_2 sharing the p first positions is adapted as follows

$$(n-d)!^m \times (n-p)^m$$

Here, the weight of a node of depth d is estimated to $(n-d)!^m$ and the number of sibling nodes in the depth d is equal to $(n-p)^m$ because we have $(n-p)$ variables to branch in each of the m permutations.

The distance in the tree decreases when d is closer to the leaves as well as the distance between the solutions in the permutation space. The inverse is not true. At the top of the tree (small values of d), two solutions are necessarily very far from each other in the tree but they could be at a minimal distance from each other according to the Hamming distance in the permutation space. Indeed, even if they diverge in the top of the tree (not the same sub-tree), they are likely to maintain the same variables in $n-2$ positions in the permutation.

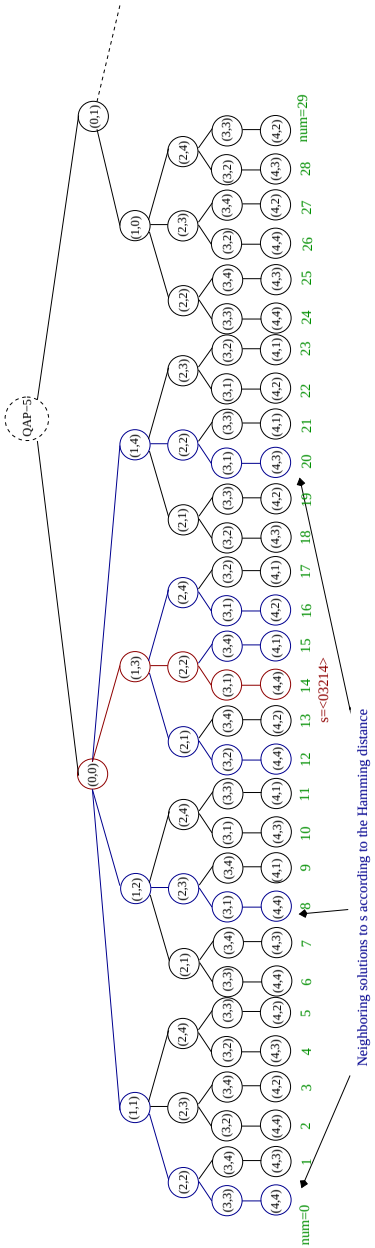


Figure 4.2: A part of the search tree for a single permutation problem (a QAP for example) of size 5 following a lexicographic order. This sub-space is represented by the interval $[0, 30[$.

4.4.2 On the connectivity of the coded tree-based representation

The deal here is to check if for every pair of solutions $\{s, s'\}$ in a given interval I there exists a path $p\{s_0, ..., s_k\}$ such as $\forall i \in \{0, ..., k\}, number(s_i) \in I$ and $s_0 = s$, $s_k = s'$.

Indeed, in case we intend to limit the search inside a given interval I, it is very important to be able to reach every solution in I starting from every other solution in I without exploring forbidden solutions whose numbers are outside I.

Nevertheless, if the order considered in the tree is lexicographic, it may happen that some sub-spaces are not reached from all the solutions because the distance between neighboring solutions in the tree does not necessarily mean they are also sharing some positions in the permutation form. A case study for such a situation is illustrated in Figure 4.3 on a QAP problem of size 5. In this example, the interval to be explored is $[6, 29]$ and the starting solution is $s =< 03214 >$ whose number is 14. If we use a 2-opt operator on this interval, only the solutions whose numbers are in the interval $[6, 23]$ could be reached during the search either from the starting solution s or from any solution in the neighborhood of s. However, none of the solutions whose numbers are in the interval $[24, 29]$ will be reached without passing through solutions which are outside of the interval to be explored.

In such case, alternatives have to be found for the connectivity issue inside the interval. In other words, in case there is some sub-regions of the interval to be explored that could not be reached through small moves, then the operator used to explore this interval should be improved with additional mechanisms such as the use of more than one move type (multiple neighborhoods), the use of multiple starting points, etc. We will limit the discussion in this section to exposing the connectivity issue in a neighborhood formed by solutions covered by the interval to be explored in a lexicographic tree. Further details on how to circumvent this limitation are given in Section 4.6.3.

In the next section, basic tools to map candidate solutions of any permutation-based problem expressed in terms of permutations, to the numbers in the lexicographic tree, are proposed. The lexicographic permutation tree is used to explain the coding and decoding operators.

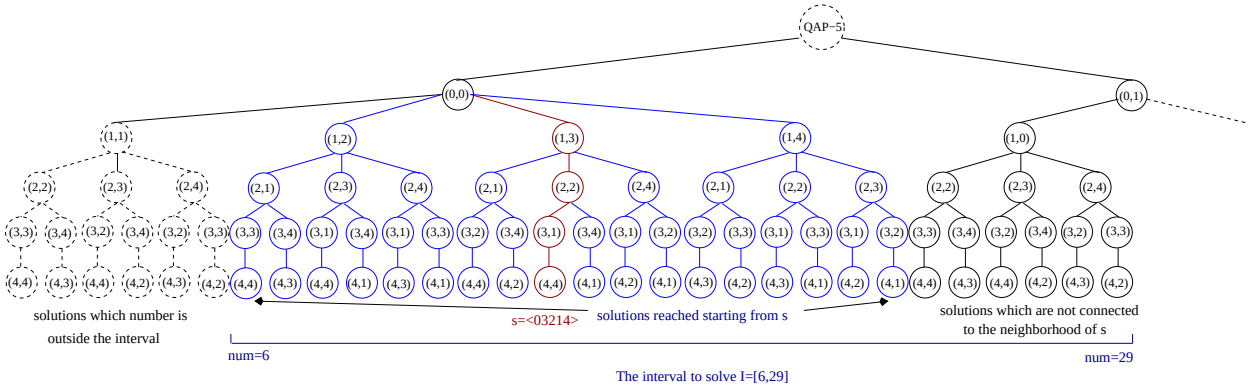


Figure 4.3: Illustration of the connectivity issue on a QAP problem of size 5. The interval to be explored here is $I = [6, 29]$, the solutions whose numbers are outside I are not considered during the search. The solutions from $num = 6$ to $num = 23$ (in blue) could be reached using a basic 2-opt move operator from the starting solution s or from any solution s' with s' in the neighborhood of s. The solutions whose numbers are in the interval $[24, 29]$ are disconnected from the neighborhood of s. Thus, they are not reached by the 2-opt operator starting from s.

4.4.3 Basic mapping tools between the space of permutations and the numbers in the lexicographic tree

If the permutation tree is constructed following a lexicographic order, the Lehmer code and factorial number system could be used to define a mapping between the solutions represented by permutations in the tree and the natural numbers in the interval $[0, n![$ (if the problem contains only one permutation), as explained in Section 3.2.5.

Nevertheless, the mapping methods from permutations to numbers and *vice versa* as reported in Section 3.2.5, are applicable only for permutation-based problems which use only one permutation to represent potential solutions. For more than one permutation, the mapping algorithms have to be adapted. On the other hand, only complete solutions (leaves in the tree) could be mapped in and back to numbers using the algorithms described in Section 3.2.5. Thus, in this section, first we propose a generalization for the mapping algorithms permutations-to-numbers and numbers-to-permutations for permutation-based problems with m permutations, $m > 0$.

Second, a new algorithm for mapping incomplete solutions (intermediate nodes in the tree), to numbers is proposed for permutation-based problems with m permutations. This method is based on the coded permutation tree as defined in [76]. The two contributions are explained in the following sections.

4.4.3.1 Mapping between solutions with m permutations and numbers

From permutations to numbers

First, let us generalize the algorithms described in Section 3.2.5 to problems with two permutations (such as Q3AP). Let s be a candidate solution for Q3AP: $s =< \phi_1, \phi_2 >$. We define the Lehmer code $L(s)$ for this solution as a couple composed of the Lehmer codes of the two permutations ϕ_1 and ϕ_2 as indicated in Equation 4.5.

$$L(S) =< L(\phi_1), L(\phi_2) > \tag{4.5}$$

With $L(\phi_1) = l_{11}d_{12}\dots l_{1n}$ and $L(\phi_2) = l_{21}d_{22}\dots l_{2n}$. The number associated to the solution s in the lexicographic order, is calculated according to Equation (4.6).

$$Number(s) = \sum_{i=1}^{n}(l_{1i} \cdot (n-i+1) + l_{2i}) \cdot (n-i)!^2 \tag{4.6}$$

If the problem uses two permutations to represent solutions, the global search space size is $|S_n| = n!^2$ for a problem of size n. Thus, the unique number associated to each solution is included in the global interval: $number(s) \in [0, n!^2[$. The term $(n-i)!^2$ in Equation (4.6) is due to the fact that at each position i in the Lehmer code from the left, the radix is $(n-i)$ and the system is factorial square because we have two permutations. If we refer to the coded tree used in the B&B algorithm, the term $(n-i)!^2$ represents the weight of a node at the depth i in the tree and the term $l_{1i} \cdot (n-i+1) + l_{2i}$ represents the rank of the node at depth i among its sibling nodes.

Now, if we consider a problem with m permutations, Equation (4.6) could be generalized as indicated in Equation (4.7) with: $number(s) \in [0, n!^m[$.

$$Number(s) = \sum_{i=1}^{n}[\sum_{j=1}^{m-1}(l_{ji} \cdot (n-i+1)^{m-j}) + l_{mi}](n-i)!^m \tag{4.7}$$

Algorithm 6 summarizes the mapping steps of a solution of a permutation-based problem represented using m permutations, into a number in the interval $[0, n!^m[$. This algorithm has a polynomial complexity.

Algorithm 6: Pseudo code that transforms a solution represented using m permutations into a number in the interval $[0, n!^m[$

Input: A solution s represented by m permutations: $\pi_1, \ldots, \pi_m$
Output: $number(s)$

Generate Lehmer codes for s: $L(\pi_1), L(\pi_2), \ldots, L(\pi_m)$
With $L(\pi_j) = l_{j1}l_{j2}\ldots l_{jn}$
`//Main loop`
for i *from* 1 *to* n **do**
    $number \leftarrow number + \sum_{j=1}^{m-1}(l_{ji} \cdot (n-i+1)^{m-j}) + l_{mi}](n-i)!^m$
return $number$

For example, let s be a solution for a Q3AP benchmark of size $n = 3$, s is represented by the two permutations

$$\begin{matrix} 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix}$$

The Lehmer code of this solution is

$$\begin{matrix} 1 & 1 & 0 \\ 2 & 0 & 0 \end{matrix}$$

The number associated to s is

$$Number(s) = (1 \cdot 3 + 2) \cdot (3-1)!^2 + (1 \cdot 2 + 0) \cdot (3-2)!^2 + 0 = 22$$

From numbers to permutations

The other way, meaning transforming a number in the interval $[0, n!^m[$ into a solution represented using m permutations, is done in two steps. First, the number is transformed to m factorial numbers of size n. The m factorial numbers are interpreted as Lehmer codes and are transformed to the corresponding m permutations. In this procedure, only the first step needs to be defined for the case of m permutations with $m > 1$.

Algorithm 7 illustrates the steps for transforming a natural number into a couple of Lehmer codes corresponding to a solution with two permutations. In this algorithm, it is

even not necessary to know the size of the permutations (or Lehemer codes) to produce. The elements of the two factorial numbers to generate are determined from left to right by a couple of components: first element of the first factorial number and first element of the second factorial number then the second elements of the two numbers and so on. The elements of the first factorial number are determined by dividing the rank[1] by the position i. In other words, if we divide the rank of the node by the position i we obtain the number of elements that are smaller than i and appear to the right of the element in position i in the first permutation. This definition corresponds to the i^{th} element from the right in a Lehmer code. The elements of the second factorial number (or Lehmer code) to generate are determined by the remainder of the caution $rank/i$. Finally, the number to be transformed is iteratively divided by the radix at each position i. The algorithm stops when $M = 0$. The final size of the Lehmer codes is known at the end of the algorithm.

Algorithm 7: Converting a natural number into two factorial numbers (or Lehmer codes)

Input: Natural number M
Output: Representation of M in Lehmer code: $l_{11} \dots l_{1n}$, $l_{21} \dots l_{2n}$

$i \leftarrow 0$
`//Start by digits from left to right`
while $M \neq 0$ **do**
  $i \leftarrow i + 1$
  $radix \leftarrow i!$[2]
  $rank \equiv M \mod radix$
  $l_{1i} \leftarrow rank/i$
  $l_{2i} \equiv rank \mod i$
  $M \leftarrow M/radix$
$n \leftarrow i$
return $(l_{1n} \dots l_{11}, l_{2n} \dots l_{21})$

Algorithm 8 is a generalization for m permutations ($m > 1$).

Once we get the m sequences $l_{1n} \dots l_{11}$, $l_{2n} \dots l_{21}, \dots, l_{mn} \dots l_{m1}$ which represent the Lehmer codes of a set of m permutations of size n, each Lehmer code is converted to the corresponding permutation of the same size using the algorithm described in Section 3.2.5.

For example, if we consider the number $M = 22$ which represents a solution $s =< \pi_1, \pi_2 >$ for a Q3AP instance of unknown size, the steps to generate the corresponding Lehmer codes $L(\pi_1), L(\pi_2)$ are illustrated in Table 4.1. When the two Lehmer codes are transformed to permutations we get the solution of the previous example

$$\begin{matrix} 2 & 3 & 1 \\ 3 & 1 & 2 \end{matrix}$$

1. which refers to the position of the current node among its sibling of the depth i in the tree

Algorithm 8: Converting a natural number into m factorial numbers interpreted as Lehmer codes for m permutations

Input: Natural number M
Output: Representation of M in Lehmer code: $l_{11} \dots l_{1n}$, $l_{21} \dots l_{2n}$

```
1  i ← 0
2  //Start by digits from left to right
3  while M ≠ 0 do
4      i ← i + 1
5      radix ← i!^m
6      rank ≡ M  mod radix
7      for j from 1 to m − 1 do
8          l_ji ← rank/i^(m−j)
9      l_mi ≡ rank  mod i
10     M ← M/radix
11 n ← i
12 return (l_1n ... l_11, l_2n ... l_21, ..., l_mn ... l_m1)
```

4.4.3.2 Mapping between incomplete solutions with m permutations and numbers

It may happen that we need to deal with incomplete solutions (solutions with incomplete permutations) or intermediate nodes in the tree representation. In [76], the numbers given to the intermediate nodes in the B&B tree are found relatively to the previous node of the same depth during the process of enumeration. However, if we intend to use the tree representation in metaheuristics, we need to recognize the number associated to any incomplete solution regardless of the previous nodes in the enumeration process, because there is no notion of enumeration in metaheuristics. In the following we adapt the previous algorithms to the case of intermediate solutions with m permutations.

From incomplete permutations to numbers

If we need to give a unique number to the intermediate nodes in the tree (meaning incomplete solutions) the same algorithms (Algorithm 6 and Algorithm 8) could be used with a small adaptation. The main loop in Algorithm6 is from 1 to n in order to calculate the number associated to a complete solution of size n. If the solution is completed only until a position p (also corresponds to the depth p in the tree), then it is sufficient to stop the calculation at the right depth (see Algorithm 9).

M	i	radix	rank	Lehmer codes
22	1	1	0	0
				0
22	2	4	2	1 0
				0 0
5	3	36	5	1 1 0
				2 0 0
0				$L(\pi_1)$ =1 1 0
				$L(\pi_2)$ =2 0 0

Table 4.1: Transformation of the number M=22 into two Lehmer codes.

Algorithm 9: Pseudo code that transforms an incomplete solution represented with m permutations into a number in the interval $[0, n!^m[$

Input: Solution s represented by m permutations of size n: $\pi_1, \ldots, \pi_m$. With s an incomplete solution: the permutations of s are completed only until the position $p < n$ (all the permutations are completed to the same position)

Output: $number(s)$

1 Generate Lehmer codes for s until the position p: $L(\pi_1), L(\pi_2), \ldots, L(\pi_m)$

2 With $L(\pi_j) = l_{j1} l_{j2} \ldots l_{jp}$

3 `//Main loop`

4 **for** i *from* 1 *to* p **do**

5 $\quad number \leftarrow number + \sum_{j=1}^{m-1} (l_{ji} \cdot (n-i+1)^{m-j}) + l_{mi}](n-i)!^m$

6 return $number$

From numbers to incomplete permutations

To generate an incomplete solution s_p with p fixed positions from its associated number $number(s_p) = M$, we need to generate the complete solution s associated to the same number M. Indeed, it is not possible to generate only a segment of the Lehmer code of the complete solution because according to Algorithm 8, the Lehmer code is constructed from the last digit to the first one (from left to right), while here we need the p first digits from the right. Thus, we use Algorithm 8 to generate the complete solution s then we determine the position p which defines s_p. For this last step, we will use the tree to illustrate the generation of an incomplete solution from its number.

Each number in a permutation tree represents a unique complete solution, or a leaf. Some intermediate nodes in the path of the considered leaf, whose number is M, are assigned the same number M. The depth d of such nodes is not known *a priori*. The question here is how

to determine the depth d of the largest tree node s_d such that

$$number(s_d) = number(s) = M \ \ and \ \ number(parent(s_d)) \neq M \tag{4.8}$$

The structure of a lexicographic permutation tree makes it that the roots and intermediate nodes of the branches in the tree which are situated at the left borders of a sub-tree, satisfy the Condition (4.8). Here we consider M as the number associated to the leaf of one of those branches. Those branches have the particularity of having 0 inversions from the depth d to the leaf. In the permutation representation, this means the segment of the permutation from position d to position n is ordered. As a result, the incomplete solution s_p having $p = d - 1$ fixed positions has the same number (M) as the leaf. In order to determine d, it is sufficient to determine, from the permutation form of the complete solution, the largest ordered segment $\pi(d), \pi(d+1), \ldots, \pi(n)$ and the incomplete solution is $s_p =< \pi(1), \ldots, \pi(d-1) >$ with $p = d - 1$.

For example, if we consider a problem with one permutation of size $n = 4$, and the solution number $M = 6$. The Lehmer code associated to this solution is 1000 and the corresponding permutation is 2134. The sequence from position 2 to the position 4 in the permutation is ordered. Thus, the largest tree-node (or incomplete solution) which respects Condition (4.8) has the depth 2 and the incomplete permutation representation is $\pi' =< 2 * ** >$ with $number(\pi') = 6$.

4.5 Interval-based initialization method for population-based metaheuristics

The first use of the numbering technique described in the previous section, consists in generating a uniformly distributed (over the tree) and diversified initial population for population-based metaheuristics. Let P be the population size used in a sequential GA. The basic idea is to split the global interval noted I, into P disjoint parts (sub-intervals). Let m be the number of permutations in the tackled problem. The global interval is $I = [0, n!^m[$. The next step is to use a pseudo-random generator to generate a unique solution in each sub-interval (of size I/P). We finally obtain a population of individuals represented only by their associated numbers. Then, to generate the solutions (represented by permutations) we use the decoding operator, from numbers to permutations, explained in Section 4.4.3.1.

Figure 4.4 illustrates a permutation tree for a one-permutation problem of size 4. In this example, the search space size is 24. The population size is fixed to 6. Then, six contiguous equal intervals are formed in order to generate one individual per interval.

To compare the complexity of the interval-based initialization method to a random initialization, we measured the execution time of the two methods on two Q3AP benchmarks of size 18 and 30. We used different population sizes. The results are shown in Table 4.2. We notice that the random initialization is at least 4 times faster than the interval initialization. However, the recorded times are still not significant ($< 1s$).

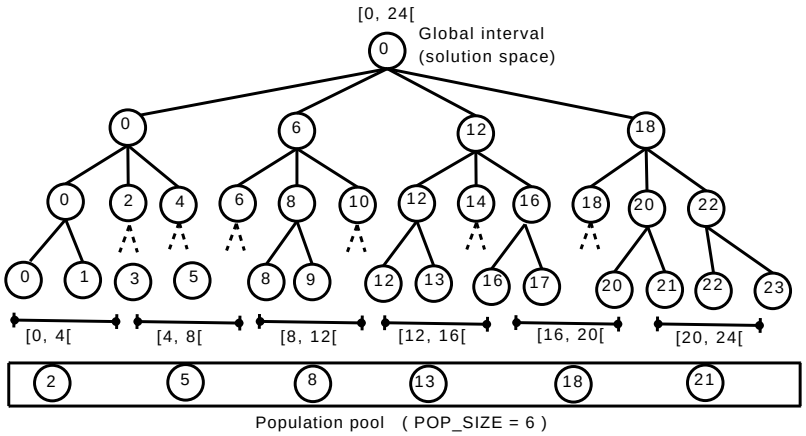


Figure 4.4: Illustration of the interval-based initialization procedure.

4.6 Interval-based search operators for metaheuristics (I-operators)

It may happen that we need to restrict the search in metaheuristics to a given sub-space. This may occur for example, in case an *a priori* knowledge about the location of the optimal solution is available, in case a part of the search space has been already searched by an exact algorithm and this sub-space may be excluded from the search, etc.

The use of the coded tree in this context might help to guide the search of metaheuristics and to restrict it to a given sub-space, represented as an interval. Mainly, the metaheuristic will be allowed to evaluate only moves which lead to solutions whose numbers are inside the interval. Special move operators taking into account the concepts of locality and connectivity should be defined in order to efficiently explore the interval at hand.

In the rest of this dissertation, we will call such operators *Interval-based operators* and we will use the notation *I-operator*. The formal definition of such operators is given in Definition 9.

Definition 9 (I-OPERATORS)**.**
An interval-based search operator, noted I-operator, defines a neighborhood of a solution s_0 with $number(s_0) \in I$, as

$$N_I(s_0) = \{s_i / number(s_i) \in I\}$$

That is, every solution s_i from the neighborhood as well as the starting solution s_0, are included in the search space delimited by the interval to be explored.

By analogy, we extend the definition of I-operators to genetic operators as well. We also define interval-based optimization algorithms, noted I-algorithms, as restricted search algorithms based on I-operators. The I-operators must be studied in terms of efficiency on

Bench.Size	Pop.Size	Algo.	INIT.Time (μs)
18	100	Rand.	2e+4
	100	Inter.	7e+4
	250	Rand.	5e+4
	250	Inter.	19e+4
	500	Rand.	7e+4
	500	Inter.	36e+7
30	100	Rand.	2e+4
	100	Inter.	8e+4
	250	Rand.	3e+4
	250	Inter.	21e+4
	500	Rand.	7e+4
	500	Inter.	42e+4

Table 4.2: Execution time of the two initialization methods random initialization and interval-based initialization.

localizing the best solution in the considered sub-space (which could be different from the global best solution of the problem), and in terms of ability to cover and to reach all the solutions included in the considered sub-space as explained in Section 4.4.2.

In the next section, we propose two types of I-operators for permutation-based problems. The first class of operators uses the tree to determine the valid moves [1]. In the following, we refer to this class of operator as *tree-based I-operators*. The second class of I-operators uses the permutation representation of the candidate solutions to determine if the move is valid or not. This class is referred to as *permutation-based I-operators*.

4.6.1 Tree-based I-operators

As explained earlier, according to the concept of locality of a representation, special care must be taken when defining the I-operators as not all the solutions included in the given sub-interval are close to each other in terms of distance between permutations, even if they are neighbors in the tree-based representation (they are in the same sub-tree). The notion of distance is very important in this context. To guarantee a high locality, the chosen distance in the tree is the Hamming distance between permutations. Consequently, the new set of search operators will continue to operate in the space of permutations and use the numbers and the tree only to avoid bad moves.

The simplest way to generate a neighborhood containing only solutions whose numbers are included in a given interval I, is to exploit the tree itself. Indeed, the positions in the permutation correspond to the depths in the permutation tree: an intermediate node in the tree with a depth d corresponds to a permutation in which only the $d-1$ first positions are

1. moves which leads to solutions whose numbers are inside the interval to be explored

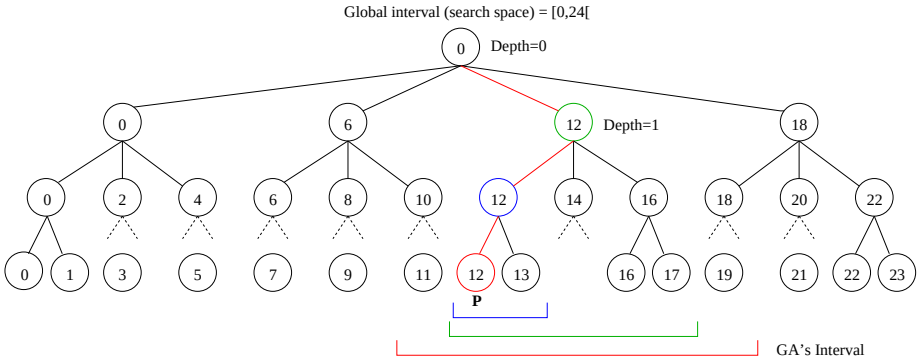


Figure 4.5: Selecting the threshold position in a given solution for the tree-based I-operators.

fixed.

If the range of an intermediate node in the tree with a depth d is included in the interval I, then all the moves done in the positions situated between the position d and the end of the permutation will automatically result in solutions belonging to the sub-tree rooted at this intermediate node. Consequently, the numbers associated to these solutions are included in the interval I by definition.

More concretely, from a starting solution s_0 (which is also a final node or a lief in the tree), we need to determine the intermediate node s_d (a node belonging to the path of s_0) for which the range is included in the interval I and the range of its parent in the tree is not in I. For this, we proceed as illustrated in Figure 4.5. First, the starting solution s_0 is mapped to the coded tree-based representation by generating its associated number in the tree. Then, the ranges associated to all intermediate tree nodes belonging to the path of s_0 are computed. For example, in Figure 4.5 for the node P, the ranges of all its predecessors, represented by the blue intervals, are computed. Next, the node whose range is included in the interval I and the range of its parent is not, is selected. In Figure 4.5 it is the predecessor of P with $depth = 1$. Then, the depth of this node is used as a *threshold position* in the permutation (see Equation (4.9)). All the moves must be performed to the right of this position in order to generate only solutions whose numbers are included in I.

$$threshold\_position(s_0) = \begin{cases} depth(n)/ \\ n \in path(s_0), \\ and \\ range(parent(n)) \nsubseteq [a,b[\end{cases} \quad range(n) \subseteq [a,b[\tag{4.9}$$

4.6.1.1 I-PMX crossover

For application, let us consider the Partially Mapped Crossover (PMX) [45]. The latter consists in randomly selecting a cutting point in the two parents. Then, the first part of the first offspring C_1 (respectively the second offspring C_2) is inherited from the

first part of the first parent P_1 (respectively the second parent P_2) and the remaining positions are completed from the appropriate alternate parents. To complete the remaining positions in the offspring (after the cutting point), genes that do not lead to conflicts (the replication of genes is not allowed) are directly inherited by the offspring from the appropriate alternate parent, while conflicting positions are filled using a special mapping relationship.

First of all, the GA is given a sub-interval representing the sub-space to be explored, that is the interval I. The initial population of GA is generated such that the numbers associated to all the individuals (or solutions) are included in I. When the operator is applied on a pair of individuals (parents), each parent is mapped to the tree-based representation by generating its associated number in the tree. Then, the threshold position in each parent is calculated. This value is used as a minimum cutting point in the PMX crossover in each parent. Let P_1 and P_2 be the two parents with the thresholds t_1 and t_2 respectively.

If the first part of the offspring is inherited from P_1 then the cutting point in the permutation is chosen randomly in the set $\{t_1, t_1 + 1, \ldots, n\}$, otherwise (if the first part of the offspring is inherited from P_2) it is randomly chosen in the set $\{t_2, t_2 + 1, \ldots, n\}$. Thus, the number associated to the resulting offspring is included in the range of one of the two parents. Consequently, it is also included in the interval I.

In the rest of this thesis, we designate this special crossover as an interval-based PMX noted I-PMX. In the context of I-PMX, this procedure is not much time consuming because the determination of the appropriate threshold position is done only once, when the initial population is given to the GA. This point is inherited by the next generations.

The same procedure can also be used for mutation operators. For example, if we consider a swap mutation operator where two points are selected at random and the genes in these positions are swapped, the threshold position in the individual to mutate is calculated and used as a lower bound for the swapping points.

4.6.1.2 Advantages and drawbacks

This type of operators has the advantage of being completely free from any additional overhead in execution time compared to their standard counterpart operators. Indeed, the operation consisting in calculating the threshold position in the solutions has a polynomial complexity and is done only once on the starting solutions (or individuals in the context of GAs).

The major drawback of the tree-based operators is the lack of diversity during the search. The search is initially restricted to the interval I but it is even more restricted because we explore only solutions belonging to the sub-tree of a single node whose range is included in the interval I. Indeed, the interval I is likely to contain several nodes with different depths in the tree but in the procedure which consists in selecting the threshold position, only one node is concerned with the operations and the remaining nodes with their sub-trees are ignored during the search. The generated neighborhood is restricted more than needed. However, if several

starting points are used (like in a GA), the effect of this drawback is blurred. Especially if the initial population of the GA is generated using the interval-based initialization method presented earlier in this chapter, as it allows to have at least one individual in each sub-tree contained in the interval to be explored.

4.6.2 Permutation-based I-operators

In this class of operators, the search is also done in the permutation space but the numbers will be used to check if the generated move leads to a valid solution, a solution whose number is included in the interval to be explored. This interval is delimited by two numbers which corresponds to two solutions in the tree (leaves). Let $I = [a, b]$ be the interval, and s_{inf}, s_{sup} be the two solutions that delimit the interval (which means $number(s_{inf}) = a$ and $number(s_{sup}) = b$). A move is valid if it is leading to a candidate solution s' whose number verifies the condition $a < number(s') < b$. There are two ways to check this condition:

- The lexicographic order relation is used on the permutation forms of the bounding solutions s_{inf} and s_{sup} and in the solution s' to determine if the condition $s_{inf} < s' < s_{sup}$ is satisfied (which means s_{inf} precedes s' and s' precedes s_{sup} in the tree following the lexicographic order).

- The number associated to the solution s' is calculated and compared to the bounds of the interval with a complexity O(n) (due to the generation of $number(s')$).

The steps to determine the order relation in a lexicographic tree between two solutions s and s' are illustrated in Algorithm 10, for a permutation problem with m permutations. This algorithm is used twice in order to determine the precedence relation in the tree between s_{inf} and s' and then between s' and s_{sup} and it does not require the calculation of $number(s')$.

If the number of permutations in the tackled problem is only one, then this method could be faster than using the numbers. But if more than one permutation are used, the complexity in the worst case is $O(m{\cdot}n)$. For example, in the case of Q3AP it is cheaper to use the numbers instead of the permutation forms. The number associated to the solution s' is calculated and compared to the two bounds of the interval (a and b) to make a decision if the move will be retained or rejected. To make this decision faster, the threshold position used in the tree-based strategy is used first. If the positions concerned by the move are situated to the right of the threshold position in the permutation, than this move is automatically accepted and there is no need to generate the solution s' or to calculate its number. Otherwise, the solution s' and its number are generated and the latter is used to decide if the move is valid or not.

4.6.2.1 Example of application: I-2opt neighborhood operator

In order to restrict the neighborhood generated by a 2-opt operator to solutions whose numbers are in a given interval I, each time a new 2-opt move is generated (in the permutation form of the solution), it is applied on a temporary solution. If the obtained solution s' satisfies the condition $s_{inf} < s' < s_{sup}$, then this move will be considered for evaluation. Otherwise, the move will be rejected and the next move is generated, and so on.

Algorithm 10: Algorithm to determine if two solutions of a permutation problem with m permutations are enumerated in a lexicographic order

Input: The two solutions s and s'
Input: With $s =< \pi_1, \ldots, \pi_m >$ and $s' =< \psi_1, \ldots, \psi_m >$
Output: A boolean indicating if s precedes s' in a lexicographic enumeration of all the permutations of size n.

```
1  //Main loop
2  i ← 1
3  while i < n do
4      j ← 1
5      while j ≤ m do
6          if π_j(i) < ψ_j(i) then
7              return TRUE
8          else
9              if π_j(i) > ψ_j(i) then
10                 return FALSE
11         j ← j + 1
12     i ← i + 1
```

Algorithm 11 is a pseudo-code of the procedure that generates the next move in the I-2opt operator (interval-based 2-opt operator).

Note that this operator could also be designed following the tree-based strategy. It is sufficient to determine the threshold point during the initialization step and the 2-opt move is applied only on positions to the right of the threshold in the permutation. In order to differentiate between the two variants of the I-2opt, in the rest of this dissertation the permutation-based I-2opt operator is noted IP-2opt and the tree-based variant is noted IT-2opt. If the IT-2opt is used, there is no need to verify if the generated moves lead to solutions inside the interval or not.

4.6.2.2 Advantages and drawbacks

Unlike the tree-based I-operators presented in the previous section, the permutation-based I-operators explore all the sub-trees that are covered by the interval I because all the moves are considered and checked one by one.

This class of I-operator could be generalized to other move operators for all permutation-based problems. However, if the operation of generating the next move is CPU time intensive in itself regardless of the interval, then the complexity of the I-operator will increase especially for large size problems.

Algorithm 11: Algorithm to generate the next I-2opt move in the neighborhood of the currant solution, inside the interval $[a, b[$

Input: The Interval $[a, b[$
Input: The initial solution s_0
Input: tr, the threshold position in s_0
Input: The last move performed at time t: m_t
Output: The next move m_{t+1}

```
1  //Main loop
2  while move not accepted do
3      m_{t+1}(i,j) ← 2opt_next_move(m_t)
4      if i ≥ tr and j ≥ tr then
5          move_accepted ← TRUE
6      else
7          s' ← apply_move(m_{t+1}, s_0)
8          number(s') ← calculate_number(s')
9          if (number(s') ≥ a and number(s') ≤ b then
10             move_accepted ← TRUE
11         else
12             move_accepted ← FALSE
13 return m_{t+1}
```

For example, if we consider the shift operator where two positions are randomly generated in the permutation and the positions between the two points are shifted to the right or the left. It is very expensive to generate all the moves and check if the generated solutions are covered by the interval I. In this case, it is better if the bad moves are automatically discarded without generating them. Thus, for the shift operator, the tree-based strategy (the use of the threshold position) is more suitable. The last issue is related to the cost of the operations of elimination of bad moves. Indeed, even if the algorithm which generates the number of the solution obtained by the actual move is done in O(n), if it is used very often during the search, it might lead to an overhead in the execution time. The effect of the operations of elimination of bad moves is studied in the experiments presented in Section 4.7.4.

4.6.3 Limitation and potential improvements of the I-operators in a lexicographic tree

As already quoted in Section 4.4.2, when designing I-operators in a lexicographic tree, special care must be given to the connectivity issue. Otherwise, it might happen that some sub-spaces in the interval remain unreachable by the I-operators. In terms of diversity of the search in a given interval, the permutation-based I-operators have a better coverage of the sub-spaces included in the interval compared to the tree-based I-operators which restrict the search to a single sub-tree whose the range of the root node is included in the interval.

Nevertheless, even in the permutation-based I-operators it might happen that some sub-trees whose ranges are in the interval are not reached without passing through solutions outside the interval, because of the lack of connectivity in the lexicographic tree in some cases (see Section 4.4.2). In order to solve this issue, we propose three potential solutions in the following.

The use of multiple starting points

The I-operators are used in a population-based metaheuristic in which the initial population is uniformly initialized from the interval. All the sub-trees covered by the interval will be represented at least by one individual in the population. Thus, all the solutions in the interval could be reached during the search by the I-operators.

The extension of the interval to be explored

The interval to be explored, let it be $I = [a, b[$, could be extended to $[a - delta, b + delta[$ in order to include those solutions which help to reach the inaccessible sub-spaces in the original interval $[a, b[$. The $delta$ should be determined according to the size of the tree nodes included in the interval $[a, b[$. Let p be the largest tree node covered by the interval $[a, b[$, with $range(p) \subset [a, b[$, then we set $delta = range(p)$.

Indeed, the inaccessible solutions in I are necessarily situated at the end of the interval or at its beginning. The structure of the lexicographic tree makes it that, at each depth of the tree, the solutions situated at the beginning of the sub-trees formed by the nodes of the same depth, are connected to each other. Which means, the Hamming distance between them is minimal. Once one of the solutions in a given sub-tree, let it be s', is reached by the search operator, the other solutions included in the same sub-tree are also reached because some of them are situated in the neighborhood of s'. For example, in Figure 4.3 the interval to be explored is $I = [6, 29]$. The solutions of the sub-interval $[24, 29]$ are not reached starting from the solution whose number is 14. If we extend the interval I from $I = [6, 29]$ to $I' = [6 - 6, 29 + 6]$, the solutions whose numbers are in the interval $[0, 6[$ are connected to the neighborhood formed by the solutions in interval $[6, 23]$. On the other hand, the solution number 0 and the solution number 24 are neighbors in the permutation space according to the Hamming distance. Consequently, if the interval $[0, 6[$ is included in the search, the solutions in $[24, 29]$ will no longer be ignored by the search. Of course, the solutions in the intervals $[0, 6[$ and $[30, 35[$ are used only to reach some isolated solutions in the original interval. Thus, it does not matter if some of them are also isolated, as they were initially outside the search.

The use of a transposition order in the tree instead of a lexicographic one

If the transposition order is used in the tree, the distance between each couple of successive solutions (having successive numbers) is equal to the minimal distance between permutations $d_{min} = 2$. Consequently, the neighborhood formed by the solutions covered by any interval, no matters its size, is always perfectly connected.

This assertion can be easily proved by induction. Let $I = [a, b]$ be an interval which covers a set of solutions enumerated in a transposition order. If $size(I) = 2$, which means I contains two successive solutions enumerated in a transposition order, then the neighborhood formed by the solutions covered by I is perfectly connected by definition. Let us suppose that for $size(I) = n$, I is perfectly connected. We need to prove that the interval $I' = [a, b + 1]$, with $size(I') = n + 1$, also forms a connected neighborhood. The solution whose number is b could be reached through some solutions in $[a, b]$ because I forms a perfectly connected neighborhood. The solution whose number is $b+1$ is at a minimal distance from number b as they are enumerated in a transposition order. Thus, $b + 1$ is connected to the neighborhood formed by the solutions in $[a, b]$. Consequently, the interval $I' = [a, b + 1]$ also forms a connected neighborhood. Which means between any couple of solutions whose numbers are included in this interval, there is a path formed by a finite number of solutions whose numbers are also all included in the interval and the distance between successive solutions in the path is equal to the minimal distance between permutations. This definitely solves the connectivity issue in the interval to be explored in a soft manner. However, we need to redefine all the mapping tools between the space of permutations and the numbers in the interval by taking into account the transposition order.

If we aim to work exclusively on final solutions covered by the interval to solve without any reference to the tree[1], we could reuse the reference algorithm for enumeration of permutations in transposition order in the discrete mathematics community, which is the Steinhaus-Johnson-Trotter algorithm [103, 54]. Using this algorithm, one could generate the next solution from the starting solution. But to map between the space of permutations and the numbers in the interval $[0, n![$, a new mechanism similar to the Lehmer code and factorial numbers, used in the mapping functions for lexicographic order, should be defined. This new mechanism is supposed to generate the i^{th} permutation in the transposition order and to deduce the number associated to any permutation during the enumeration process. To our knowledge, such mechanism does not exist yet in the discrete mathematics community.

Furthermore, if we want the set of all possible permutations enumerated in a transposition order, to be also arranged to form a permutation tree (in the case we intend to use some tree-based search), even the algorithm of Steinhaus could not be used because the solutions produced by this algorithm could not be arranged in a permutation tree. Thus, in all cases we need to define two new mechanisms from scratch:

- The generation of a permutation tree in which the final nodes (solutions) are enumerated in a transposition order.
- A coding/decoding mechanism which maps between a set of permutations enumerated in a transposition order and the numbers from 0 to $n!$.

That is not all, since we are interested in all permutation-based problems no matters the number of permutations used, we also need to generalize the solutions to permutation problems with any number of permutations. The idea of a permutation tree in which

1. for example, in the case we want to explore a given interval only by metaheuristics and local search methods, no reference to the tree is needed

the solutions are enumerated in transposition order is illustrated intuitively in Figure 4.6. We reuse the previous example (Figure 4.3 in Section 4.4.2) which illustrates a part of a permutation tree for a QAP problem of size 5, but the solutions are enumerated in a transposition order instead of lexicographic order. This means that the interval to solve $I = [6, 29]$ in the lexicographic tree does not contain the same solutions as in the transposition tree. Notice that the Hamming distances between any couple of successive solutions in the new tree (final nodes in the tree) is equal to the minimal distance between permutations.

Although this solution to the connectivity issue in the I-operators is very attractive and it can even be useful for other applications, we did not go further for it during this thesis. Thus, we point it out as a future work of this thesis. In the next section, the I-operators proposed in this chapter are applied to one of the hardest permutation-based problems, the 3D quadratic assignment problem (Q3AP).

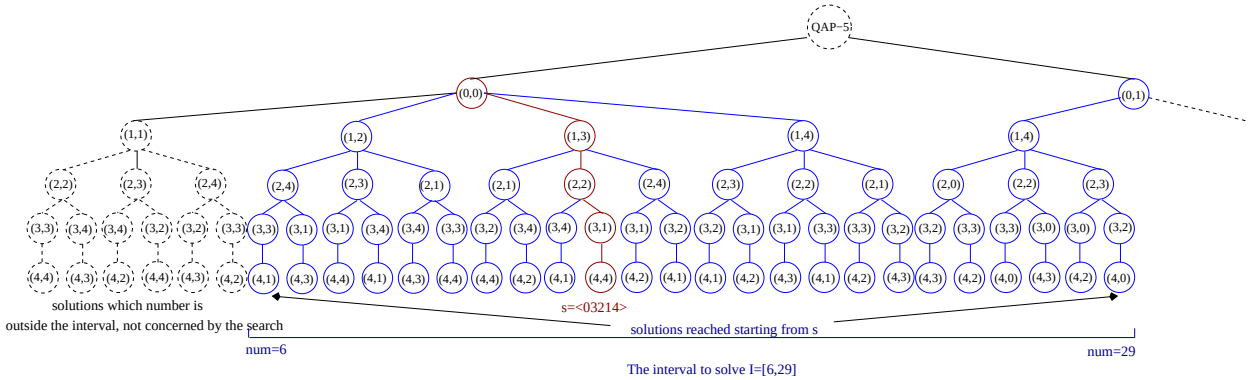


Figure 4.6: Illustration of a permutation tree in which the final nodes (the solutions) are enumerated following a transposition order. In this example, a part of the search space of a QAP problem of size 5 is shown. The interval to solve here is $I = [6, 29]$, the solutions whose numbers are outside I are not considered by the search. The solutions from $num = 6$ to $num = 29$ (in blue) could all be reached by a basic I-2opt move operator from the starting solution s or from any solution s' with s' in the neighborhood of s.

4.7 Application to Q3AP

In order to evaluate the performances of the I-operators in finding local optima inside a given interval, they were implemented for Q3AP and used in three algorithms: Hill Climbing (I-HC), Iterated local search (I-ILS) and Genetic Algorithm (I-GA). The three algorithms are explained and experimented in the following.

4.7.1 Interval-based Hill Climbing algorithm for Q3AP (I-HC)

Hill Climbing is a very basic local search algorithm with two components: the move operator which is used to iteratively modify the current solution in order to improve the actual best found solution, and the selection of the next solution to exploit in the current neighborhood. For Q3AP we used the 2-opt move operator on the two permutations and the best fitness improvement selection strategy. This means all the solutions in the examined neighborhood are evaluated and the best solution is selected to be exploited in the next iteration until no further improvement is possible. This is a standard HC algorithm for Q3AP.

In the interval-based version of the algorithm referred to as I-HC, the I-2opt is used instead of 2-opt and the initial solution is randomly generated but with the condition $number(s_0) \in [a, b[$, with $[a, b[$ the interval to be explored. This basic algorithm is implemented in order to observe, more closely, the behavior of the I-2opt operator. A pseudo code of I-HC is given in Algorithm 12.

Algorithm 12: Pseudo code of the I-HC algorithm

Input: The interval $[a, b[$
Output: Best found solution s
`//Initialization step`
Input: Randomly generate initial solution s_0 such that: $number(s_0) \in [a, b[$
`//Save initial solution`
$s \leftarrow s_0$
`//Main loop`
while *stopping criterion not reached* **do**
    $Generate - I - 2opt(N(s))$
    $Evaluate(N(s))$
    $s' \leftarrow Select(N(s))$
    $s \leftarrow s'$
return s

4.7.2 Interval-based Iterated local search algorithm for Q3AP (I-ILS)

The ILS algorithm is a stochastic search algorithm based on three essential components: *local search*, *perturbation* and *acceptance criterion*. The local search could be any single solution metaheuristic. Perturbation is preferred to be a neighborhood operator that is

different from the one used in the local search component. In the basic ILS algorithm for Q3AP we used the following components:

- **Initialization step:** the starting point s_0 is randomly generated.
- **Local search:** the $2-opt$ move operator is used on the two permutations of a Q3AP solution.
- **Acceptance criterion:** once a new local optimum is found by the local search step, an acceptance criterion is used to determine if the perturbation will be applied to the new found local optima or to the previous one if any. The acceptance criterion used in the basic ILS algorithm is the one proposed in [49]. The new optimum is accepted for perturbation if its fitness is better than the current best solution. Otherwise, it is accepted with a probability 0.5 if its fitness is at most 5% worst than the fitness of the current best solution (see Equation (4.10)).

$$accept(s', s) = \begin{cases} fitness(s') > fitness(s) \ \ or \\ fitness(s') - 0.05 * fitness(s) > fitness(s) \end{cases} \tag{4.10}$$

- **Perturbation:** the shift operator is used on the two permutations of a Q3AP solution to create a perturbation. Two random points $p1$, $p2$ are chosen in the two permutations and all the positions between them are shifted to the right or to the left.

That was an ILS algorithm for Q3AP that aims to find a global optimum. If the search is restricted to an interval $[a, b[$, the initialization, the local search and the perturbation steps have to be replaced by their interval-based versions as follows:

- **Initialization:** the first solution s_0 is randomly generated but its associated number has to be inside the interval $[a, b[$: $number(s_0) \in [a, b[$. Thus a random number is generated in the interval $[a, b[$, then the decoding operator is used to deduce the permutation form of the solution s_0 from its associated number.

- **Local search:** the I-2opt neighborhood operator described in the previous section, is used instead of the 2-opt operator. Two versions of the I-ILS noted IP-ILS and IT-ILS are experimented. IP-ILS uses IP-2opt and IT-ILS uses IT-2opt.

- **Perturbation:** a tree-based I-shift operator (IT-shift) is used on the two permutations of a Q3AP solution but the two random points $p1$, $p2$ are chosen such that $p1 > threshold\_position(s_0)$ and $p2 > threshold\_position(s_0)$. With $threshold\_position(s_0)$ is the depth of the parent of s_0 with the largest range included in the interval $[a, b[$ (as previously explained).

Algorithm 13 is a pseudo-code of the I-ILS algorithm.

The use of these two I-operators in the ILS algorithm and the generation of the starting solution inside the interval $[a, b[$ will guarantee that the search will focus only on the set of solutions whose numbers are included in $[a, b[$. Nevertheless, as pointed out in Section 4.6.3, depending on the starting solution, some sub-spaces in the interval may be unreachable without passing by forbidden moves leading to solutions whose numbers are outside the

Algorithm 13: Pseudo code of the I-ILS algorithm

Input: The interval $[a, b[$
Output: Best found solution s
`//Initialization step`
Input: Randomly generate initial solution s_0 such that: $number(s_0) \in [a, b[$
`//Save initial solution`
$s \leftarrow s_0$
$s' \leftarrow s$
`//Main loop`
while *stopping criterion not reached* **do**
  $s \leftarrow local\_search(s, [a, b[)$
  if $accept(s, s')$ **then**
    $s' \leftarrow s$
  else
    `//Back to the previous solution`
    $s \leftarrow s'$
  $s \leftarrow perturbation(s, a, b)$
return s

interval $[a, b[$. Such a limitation will decrease the efficiency of the I-ILS in localizing the best solution in the considered sub-space. To encompass this connectivity issue, a I-GA is implemented.

4.7.3 Interval-based genetic algorithm for Q3AP (I-GA)

The diversification skills of GAs makes them a perfect candidate to make an efficient search inside a given sub-space delimited by an interval $[a, b[$. In order to guarantee that the search of the GA will be (1) concentrated inside the sub-space delimited by the interval $[a, b[$ and (2) that all the potential solutions of that sub-space could be reached during the evolution process, the GA is enhanced with interval-based transformation operators and the interval-based initialization method. The different components of I-GA are explained in the following.

- **Initialization:** the initialization of the first population of I-GA is done according to the uniform interval-based initialization method already explained in Section 4.5. Mainly, each individual g in the population satisfies the following condition: $number(g) \in [a, b[$. After this initialization step, each individual is mapped to the tree by its associated number. Then, the depth of the largest node belonging to its path in the tree and whose range is included in $[a, b[$, is calculated. This depth is associated to the individual and is to be used by the transformation operators as a threshold position to control the search inside the sub-space delimited by the interval $[a, b[$.

- **Selection:** the Roulette Wheel selection is used with elitism in order to save the best

individual in the population.

- **Crossover:** the used crossover is the I-PMX operator explained in Section 4.6.1.

- **Mutation:** in order to enhance I-GA with intensification skills, the mutation operator is replaced by a combination of two operators: I-ILS and the operator I-shift. This kind of hybridization is very common in the literature as it has been pointed out by many authors that GAs when combined with local search algorithms outperforms GAs alone when applied to combinatorial optimization problems. In this combined mutation, each part of the combination is applied with a fixed rate on each selected individual for mutation. The rate used is 50% for each operator. Indeed, the use of a local search only could lead to a premature convergence. Therefore, the use of a second mutation operator will help to introduce diversity in the population.

In our experiments, we consider two versions of I-GA following the type of I-ILS used in the mutation operator: IP-ILS (using a permutation-based I-2opt operator as a local search) or IT-ILS (using the tree-based I-2opt operator). To differentiate between the two implementations of I-GA , we refer to the first one as IP-GA and to the second as IT-GA.

4.7.4 Experimental results

The objective of the following experiments is to check if the I-operators will be able to efficiently explore the sub-space delimited by a given interval. The Q3AP benchmarks used are all derived from standard QAP benchmarks which could be found in the QAPLIB [14]. Some benchmarks are generated from other instances of larger size in the QAPLIB: Nug12c is generated from Nug13 in the QAPLIB (one column and one row form each matrix of Nug13 are withdrawn), Had12b is obtained after permuting the two matrices of Had12, Had15 is generated from Had16 and Nug15a from Nug16a (one column and one row from each matrix of Had16 and Nug16a respectively, are withdrawn).

In the first set of experiments, we consider only averaged benchmarks: Nug12, Nug12c, Had12, Had12b and Nug13. The following algorithms are used to solve each instance and/or interval: HC, I-HC, ILS, IP-ILS, IT-ILS, GA, IP-GA, IT-GA. For each benchmark, we consider the interval $[0, n!^2/10[$, with n the size of the benchmark. The best solution whose number is included in this interval is found (local optima) for each benchmark using a B&B algorithm for Q3AP implemented in the BOB++ framework, kindly provided by Galea and Le Cun [40].

The global optimum of each benchmark is also calculated using the same algorithm. The local optimal solution is used to evaluate the efficiency of the considered interval-based metaheuristics while exploring the interval. In Table 4.3 the benchmarks are listed as well as the global optimum and the local optimum in the interval $[0, n!^2/10[$. Unless for the benchmark Nug12, all the local optima found in the considered interval are different compared to the global optimal.

Each algorithm is run 30 times on each benchmark. I-HC, IP-ILS, IT-ILS, IP-GA and

Bench.	Size	Global optimal	Interval	Local optimal
Nug12	12	580	$[0, 12!^2/10[$	580
Nug12c	12	1326	$[0, 12!^2/10[$	1338
Had12	12	19430	$[0, 12!^2/10[$	19706
Had12b	12	21128	$[0, 12!^2/10[$	21514
Nug13	13	1912	$[0, 13!^2/10[$	1918

Table 4.3: Local and global optima for the Q3AP benchmarks Nug12, Nug12c, Had12, Had12b and Nug13.

IT-GA explore only the solutions whose numbers are included in the interval $[0, n!^2/10[$, while their standard counterparts algorithms HC, ILS and GA explore the entire search space to find the global best solution. In the following, the experiments are presented in two groups:

- Single solution metaheuristics which include HC, I-HC, ILS, IP-ILS and IT-ILS.
- Population-based metaheuristics which include GA, IP-GA and IT-GA.

4.7.4.1 Single solution algorithms

Because the objective of the experiments is to measure the impact of the process of restriction of the search in the interval-based algorithms on the quality of the solution found and on the execution time, the stopping criterion for the three versions of the ILS algorithm is the number of iterations which is fixed to 2000.

The results obtained using the algorithms HC, I-HC, ILS, IP-ILS and IT-ILS on the Q3AP benchmarks Nug12, Nug12c, Had12, Had12b and Nug13 are shown in Table 4.4, Table 4.5, Table 4.6, Table 4.7 and Table 4.8 respectively.

In the result tables, the best found solutions over the 30 runs are given for each algorithm (column 2) and average solutions among the 30 runs are given in column 4. Column 3 gives the percentage of runs in which the algorithms reach the best known solution of the benchmarks. In column 6, the number of function evaluations is given for each algorithm while in column 7 we have the percentage of rejected solutions (because they were outside the interval) in the I-algorithms. This percentage is calculated relatively to the number of evaluations. Knowing that only accepted solutions (whose numbers are inside the interval) are evaluated, the rejected solutions are not considered for evaluation to save the time. This metric is very important. Indeed, if the percentage of rejected solutions is very high this means that the search operators used in the I-algorithms are not suitable to the representation based on numbers and intervals, the visited neighborhood is tightened and this will most probably lead to poor solutions in the considered interval.

To analyze the results following the different metrics and considerations: time, percentage of rejection, solution quality, we will separate the results on three groups of questions:

1. What is the impact of the search restriction on the I-algorithms in terms of execution

time compared to their standard versions?

2. What is the quality of the final solutions found by the IP-ILS, IT-ILS relatively to the local best solution in the interval $[0, n!^2/10[$ in each benchmark?

3. What is the relation between the percentage of rejection, the size of the interval and the execution time required in the two algorithms IP-ILS and IT-ILS?

The three questions are answered in the following and additional experiments are done when necessary.

Algo.	**Best**	**Hits(%)**	**Avrg.fit.**	**SD**	**Nb. eval.**	**% rejects**	**Time (s)**
HC	904	0	1217	170.13	1889	0	0.54
I-HC	1024	0	1285	177	1608	1.2	0.45
ILS	580	29.41	674	85.65	3211016	0	828
IP-ILS	580	13.33	760	86.61	2557040	1.38	547
IT-ILS	638	0	796	69.98	2532123	0	540

Table 4.4: Results obtained using HC, I-HC, ILS, IP-ILS and IT-ILS on the Q3AP benchmark Nug12.

Algo.	**Best**	**Hits(%)**	**Avrg.fit.**	**SD**	**Nb. eval.**	**% rejects**	**Time (s)**
HC	1720	0	1944	220	2145	0	0.55
I-HC	1492	0	1963	279.97	1848	1.54	0.47
ILS	1326	51.61	1361	57.29	3345111	0	835
IP-ILS	1338	8.33	1408	60.60	2547768	1.14	555
IT-ILS	1338	8.33	1452	98.03	2650725	0	550

Table 4.5: Results obtained using HC, I-HC, ILS, IP-ILS and IT-ILS on the Q3AP benchmark Nug12c.

Algo.	**Best**	**Hits(%)**	**Avrg.fit.**	**SD**	**Nb. eval.**	**% rejects**	**Time (s)**
HC	19546	0	20717	420.45	4113	0	0.65
I-HC	20030	0	20780	738.82	5290	1.02	0.53
ILS	19764	0	20337	325.58	3601133	0	910
IP-ILS	21672	0	22041	327.63	2796001	1.4	542
IT-ILS	19922	0	20869	496.75	2798445	0	586

Table 4.6: Results obtained using HC, I-HC, ILS, IP-ILS and IT-ILS on the Q3AP benchmark Had12.

Algo.	Best	Hits(%)	Avrg.fit.	SD	Nb. eval.	% rejects	Time (s)
HC	21520	0	22358	680.98	2607	0	0.65
I-HC	22512	0	23231	656.57	2156	1.30	0.55
ILS	21128	3.85	21872	308.48	3260876	0	830
IP-ILS	21556	0	22041	327.63	2573351	1.4	542
IT-ILS	21612	0	22810	617.38	2597265	0	546

Table 4.7: Results obtained using HC, I-HC, ILS, IP-ILS and IT-ILS on the Q3AP benchmark Had12b.

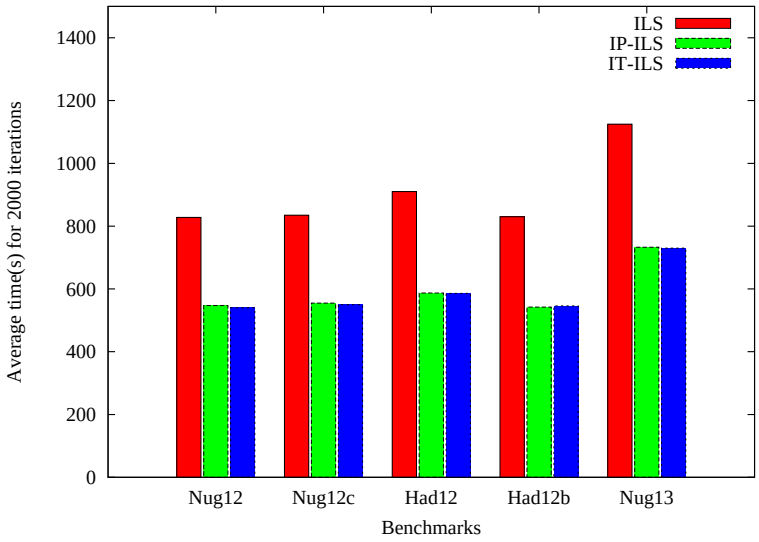
Algo.	Best	Hits(%)	Avrg.fit.	SD	Nb. eval.	% rejects	Time (s)
HC	2308	0	2915		2477	0	0.75
I-HC	2468	0	2988		2090	0.96	0.66
ILS	1974	0	2102	83.92	4349148	0	1125
IP-ILS	1918	3.85	2174	156.63	3480617	0.98	733
IT-ILS	2008	0	2127		3492736	0	730

Table 4.8: Results obtained using HC, I-HC, ILS, IP-ILS and IT-ILS on the Q3AP benchmark Nug13.

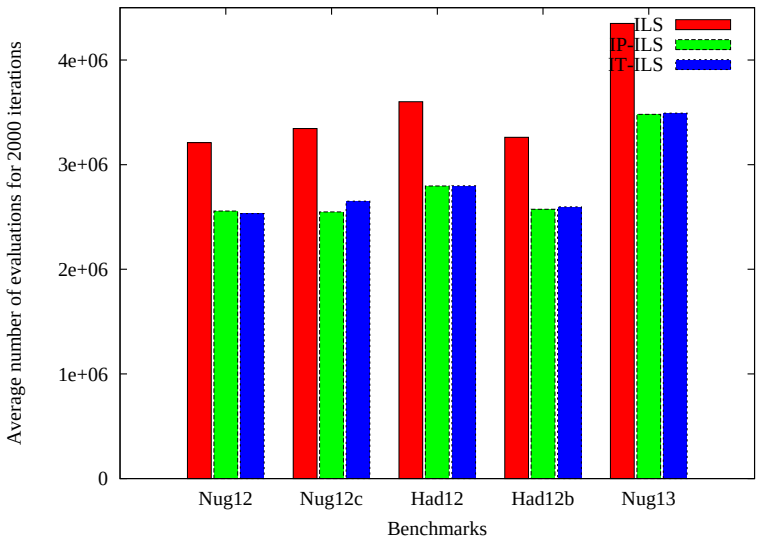
Impact of the I-operators on the execution time of the I-algorithms

First of all, if we look at the number of evaluations in the I-algorithms I-HC, IP-ILS and IT-ILS, we notice that the latters perform less evaluations comparing to their standard counterpart HC and ILS. In HC, the number of evaluations is on average 1.2 times more than the number of evaluations performed by I-HC and the time took by HC is also 1.2 times more than the execution time required by I-HC. I-HC is considered as an elementary algorithm where the I-2opt operator is used directly. In order to do further analysis the focus is set on the three versions of ILS: ILS, IP-ILS and IT-ILS and the case of I-GA will be studied later in this section.

Even if the number of iterations of the three ILS algorithms is the same, the number of evaluations and the time took for the 2000 iterations vary from one algorithm to the other (see Figure 4.7 (a) and Figure 4.7 (b)). The variation in the number of evaluations is justified by the search strategy in each algorithm. In ILS, all the moves are evaluated and took into consideration. In IP-ILS, each time a new move is generated in the local search step, it is checked if the solution produced has a number included in the interval. If it is, the move is evaluated, otherwise the move is eliminated, the next move is generated and so on. In the case of IT-ILS, the search is even more restricted: the local search operator in IT-ILS does not even consider moves acting in the left part of the solution. In other words, the moves that modify a position (in the two permutations) that is situated to the left of the minimum catting point, are automatically ignored (they are not generated).



(a) Average time took by the algorithms ILS, IP-ILS and IT-ILS for 2000 iterations on all the benchmarks.



(b) Average number of evaluations after 2000 iterations done by the three versions of ILS: ILS, IP-ILS and IT-ILS on all the benchmarks.

Figure 4.7: Statistics of ILS, IT-ILS and IP-ILS.

For example, the standard ILS algorithm performs 1.25 times more evaluations than IP-ILS and IT-ILS in the case of the benchmark Nug12 (see Figure 4.7 (a)). Consequently, the execution time of the standard ILS algorithm is also higher than the execution time of IP-ILS and IT-ILS. For example, if we consider again the benchmark Nug12, the two algorithms IP-ILS and IT-ILS achieve a speed up of approximately 1.5 compared to the standard ILS (see Figure 4.7 (b)) on the same number of iterations (2000).

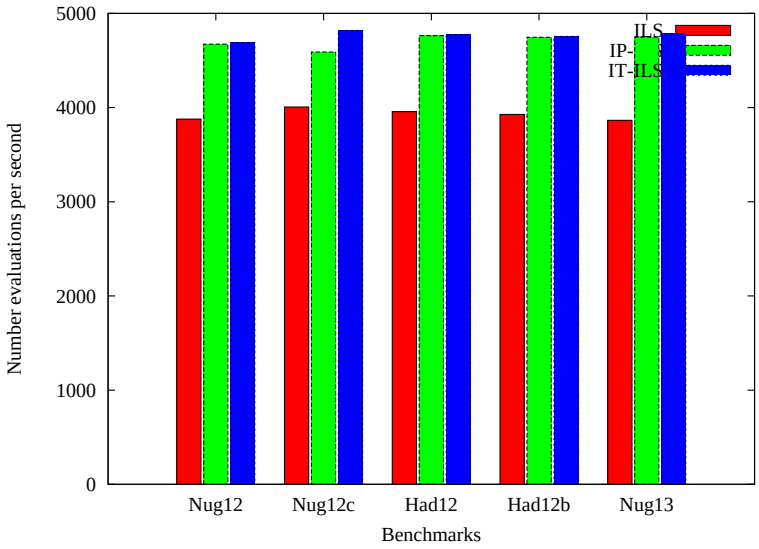


Figure 4.8: Average number of evaluations per second done by the algorithms ILS, IP-ILS and IT-ILS on all the benchmarks.

On the other hand, the average number of evaluations per second for each algorithm through the benchmarks is calculated and shown in Figure 4.8. We notice that the I-algorithms achieve more evaluations per second compared to the standard ILS. This can best be explained by the fact that the complexity of the operator used in the perturbation step of the I-algorithms is reduced because of the use of the minimum cutting point. Indeed, if one of the two points used in the shift operator is located on the left of the minimum point in the permutation, the smallest one is replaced by the minimum cutting point. Should this happen very often during the 2000 iterations, the average size of the portion to shift in the permutations will be smaller than in the standard shift operator used in the perturbation step of the ILS.

Finally, as already said, in IT-ILS, there is no extra operations compared to ILS as the search is restricted automatically by fixing the minimum cutting point in the permutations which guarantee that the search will stay in the interval without any additional verification. Thus, we use IT-ILS to measure the impact, in terms of execution time, of the operations of elimination of bad moves used in IP-ILS. From the results, we notice that the two algorithms IT-ILS and IP-ILS perform equally according to the metric number of iterations per second in all the benchmarks for the interval $[0, n!^2/10[$. This means that there is no overhead in execution time specially due to the operations of elimination of bad moves in IP-ILS for this interval size.

Impact of the interval-based operators on the final solution quality achieved using IP-ILS and IT-ILS

The optimal solution in the interval $[0, n!^2/10[$ for each benchmark is given in Table 4.3. Unless for the benchmark Nug12, the local optimal solution in all the benchmarks is not the global optimal of the problem. Thus, the solution quality of IP-ILS and IT-ILS is compared only to the best solutions in the interval.

From the results, it is noticed that IP-ILS slightly outperforms IT-ILS according to the metric average fitness on all the benchmarks. This is explained by the fact that IP-ILS explores more solutions than IT-ILS as noticed in the previous section on time analysis. However, this result is not confirmed by the Wilcoxon test as shown in Table 4.9. The Wilcoxon test is used here to compare the average fitness of the two I-algorithms. The H_0 hypothesis is: "the means of the the two populations are equal". The two populations here are made of the 30 final solutions obtained by each algorithm. The alternative hypothesis H_1 is: "the population mean obtained by IP-ILS is better than the one obtained by IT-ILS". Following this test, a result is called statistically significant if p-value < 0.05. This means that the null hypothesis is rejected and the alternative one is true. The values obtained for all the benchmarks, unless Had12 and Had12b, indicate that none of the two I-algorithms is superior compared to the other.

bench.	**p-value (IP-ILS vs. IT-ILS)**
Nug12	0.05456
Nug12c	0.1848
Had12	0.04384
Had12b	0.0001801
Nug13	0.7084

Table 4.9

Globally, the two I-algorithms IP-ILS and IT-ILS achieve good final solution quality compared to the standard ILS algorithm and relatively to the best solution in the considered interval. Indeed, the optimal solution in the interval is found in 13.33% of the runs using IP-ILS (29.41% of the runs using ILS) for the benchmark Nug12. The optimal solution in the considered interval for Nug12c is found by both IP-ILS and IT-ILS in 8.33% of the runs. For the benchmarks Had12 and Had12b, the optimal solution is not reached by the algorithms IP-ILS and IT-ILS. The relative errors measured from the average fitness achieved using the two algorithms IP-ILS and IT-ILS on the benchmark Had12 are estimated to 11.85% and 5.90% respectively. For the benchmark Had12b, the relative error achieved using IP-ILS is estimated to 0.19% and the relative error achieved using IT-ILS is estimated to 0.45%.

Evolution of the percentage of rejected moves when the size of the interval is reduced

It is interesting to study the evolution of the percentage of rejected moves when the size of the interval to be explored is reduced. For that, additional experiments were performed in the intervals: $[0, n!^2/20[$, $[0, n!^2/200[$, $[0, n!^2/1000[$, $[0, n!^2/2000[$ and $[0, n!^2/3000[$. The used benchmarks are Nug12, Nug13, Had14 and Nug15. Different problem sizes are used in this step because the size of the interval is estimated relatively to the size of the benchmark. The results are shown in Figure 4.9. In the first interval $[0, n!^2/10[$, the percentage of rejected moves in all the benchmarks is less than 2% (less than 1% for the benchmarks of size 14 and 15). As the interval is defined relatively to the size of the benchmark n, its size is different from one benchmark to the other. This explains the large difference between the percentages obtained with Nug12 and the one obtained with Nug15 for example. However, generally the IP-ILS has the same behavior on all benchmarks: when the size of the interval is reduced, the percentage of rejected moves is increasing linearly.

These results are concerned with the property of locality of the coded tree-based representation in metaheuristics. If the interval is big enough, the percentage of moves that lead to solutions outside the interval is small and does not influence the search neither in term of execution time nor in term of quality of solutions (relatively to the best solution in the interval). Nevertheless, the percentage of rejected moves relatively to the number of evaluations is increasing when the size of the interval is decreased and it can reach 50% (which means there is exactly the same number of rejected moves than accepted ones). As a conclusion, when the size of the interval is small enough, it is better to do an explicit or implicit enumeration than to use a search operator in that interval.

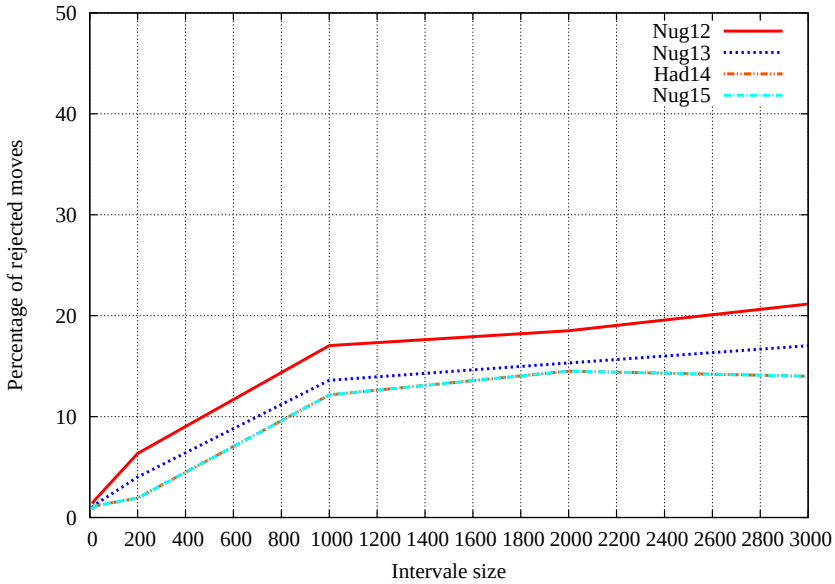


Figure 4.9: Average percentage of rejected moves in IP-ILS on all the benchmarks after 2000 iterations on different interval sizes.

Evolution of the average execution time of the I-algorithms IP-ILS and IT-ILS when the size of the interval is reduced

In this section, we focus on the evolution of the execution time of the I-algorithms IP-ILS and IT-ILS when the size of the interval is reduced. The average execution time on the different benchmarks (Nug12, Nug13, Had14 and Nug15) are calculated for the three ILS algorithms and shown in Figure 4.10. The execution time of the standard ILS is always the same as the search is global. The execution time of IP-ILS and IT-ILS is slightly decreasing when the size of the interval is also decreased. This is actually a good feature, as it shows that there is no extra overhead in execution time when using the I-algorithms on more and more restricted sub-spaces.

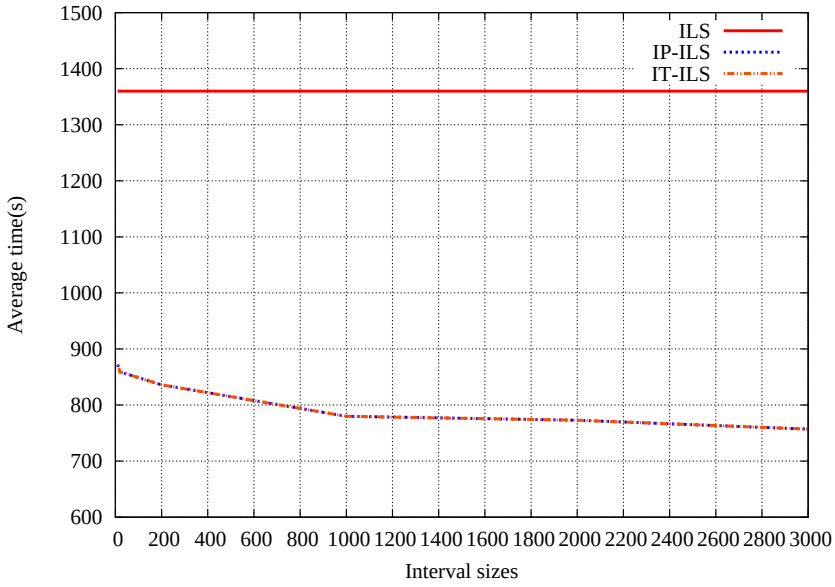


Figure 4.10: Variation of average execution times of the algorithms ILS, IP-ILS and IT-ILS on all the benchmarks after 2000 iterations on different interval sizes.

4.7.4.2 Population-based metaheuristics

The settings used in the three versions of the GA (GA, IP-GA and IT-GA) are given in Table 4.10.

Pop.Size	**Cross.Rate**	**Mut.Rate**	**Selection**
60	1	0.4	Roulette wheel
Swap-Mut-Rate	**LS.Mut.Rate**	**LS. Iter**	**Max.Gen**
0.5	0.5	4	600

Table 4.10: Global settings used for GA, IP-GA and IT-GA.

The parameters Swap-Mut-Rate and LS-Mut-Rate indicate the probability of application

of the swap and the Local search (I-ILS in our case) respectively, in the combined mutation. The parameter LS.Iter indicates the number of iterations used in the I-ILS embedded inside the mutation operator. The stopping criterion of the three algorithms is the number of generations, 600 generations as indicated in Table 4.10. The results obtained using the three algorithms are shown in Table 4.11.

For each algorithm, the best found solution, the average fitness values over 30 runs, the standard deviation of the fitness and the average execution time are shown in the columns B.Fit, A.Fit, SD and Time respectively. The measured time in seconds is the real time of the algorithms on the same dedicated machine. The last column of the table (p-value) is the result of a Wilcoxon test which compares the means of the two I-algorithms IP-GA and IT-GA. The H_0 hypothesis is: "the means of the two populations are equal". The two populations here are made of the 30 final solutions obtained by each algorithm. The alternative hypothesis H_1 is: "the population mean obtained by IP-GA is less than the one obtained by IT-GA". In the following, the obtained results are analyzed according to two metrics: execution time and quality of the final solutions.

Bench.	GA				I-GA1				I-GA2				p-value
	B.FIT	A.Fit.	SD	Time	B.Fit.	A.Fit.	SD	Time	B.Fit.	A.Fit.	SD	Time	I-GA1 vs I-GA2
Nug12	580	639	79	2540	580	674	83	3959	580	730	61	2060	0.01
Nug12c	1326	1342	17	2607	1338	1363	25	4401	1338	1394	41	2038	0.0086
Had12	19430	19430	0	2869	19706	19716	40	4601	19706	19763	82	2161	0.02418
Had12b	21128	21128	0	2531	21514	21525	19	4012	21514	21548	72	1994	0.3816
Nug13	1918	1985	25	3698	2030	2067	46	5684	1998	2047	33	2872	0.8718
Had14	37598	37613	41	5030	37746	37877	116	7836	37746	37938	234	3951	0.3802
Nug15	2230	2437	426	5485	2426	2531	70	9136	2382	2560	88	4595	0.2743

Table 4.11: Average fitness and execution time achieved by the algorithms GA, IP-GA and IT-GA over 30 runs on the Q3AP benchmarks Nug12, Nug12c, Had12, Had12b, Nug13, Had14 and Nug15.

Impact of the I-operators on the execution time of the I-algorithms IP-GA and IT-GA

The average execution time performed by each algorithm through the generations on all the benchmarks are calculated and shown in Figure 4.11. From the results in Table 4.11 and the illustration depicted in Figure 4.11, we notice that unlike IP-ILS which performs better in execution time compared to the standard ILS algorithm, IP-GA is almost two times slower than the standard GA. Conversely, IT-GA performs better than GA and IP-GA on all the considered benchmarks.

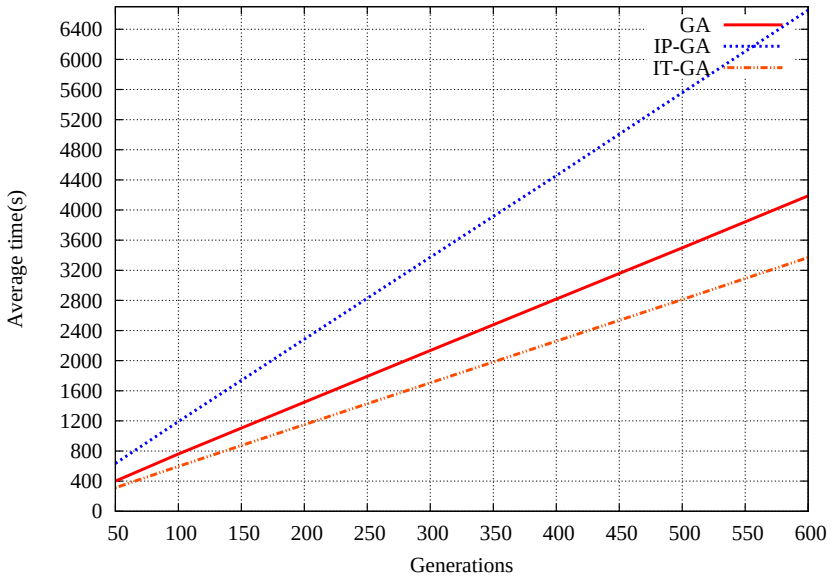


Figure 4.11: Average execution time (s) of GA, IP-GA and IT-GA on the benchmarks Nug12, Nug12c, Had12 and Had12b, Nug13, Had14 and Nug15.

Impact of the I-operators on the final solution quality in IP-GA and IT-GA

In order to evaluate the solution quality achieved by the algorithms IP-GA and IT-GA we refer to the average fitness obtained for each benchmark (see Table 4.11). IP-GA outperforms IT-GA for all the benchmarks unless for Nug13 according to this metric. However, this result is confirmed by the p-values obtained by the Wilcoxon test only in the case of the benchmarks Nug12, Nug12c and Had12. For the benchmarks Had12b, Nug13, Had14 and Nug15, the superiority of the average fitness obtained by IP-GA on the one obtained by IT-GA, is not statistically significant. This is actually a good new as from the execution time point of view, IT-GA is much faster than IP-GA and than the standard GA as well, for approximately the same performance in terms of solution quality. Thus, IT-GA is efficient enough to be used in a cooperative hybrid search algorithm combining GAs with B&B (see Chapter 5).

4.8 Conclusion

The use of numbering of permutations to represent candidate solutions of permutation-based problems in metaheuristics, is a new idea used recently by very few researchers. Coding and decoding operators based on the properties of permutations, such as Lehmer codes, and factorial number systems are used to switch between the two representation spaces: permutations and numbers. Generally, the numbers are directly used by the search operators while the permutation representation is used to compute the fitness value. In this chapter, coding and decoding tools are proposed for permutation-based problems using more than one permutation to represent solutions. These tools are also based on factorial number systems and Lehmer codes. Permutation trees (used in some exact methods such as B&B) are also used for more understanding of the coding/decoding mechanism. This new representation could have different uses in metaheuristics. In this thesis, it is mostly used to restrict the search of the metaheuristic to a given sub-space represented as an interval containing the numbers associated to all the solutions in the sub-space to be explored. Nevertheless, as the concepts of locality of representation and connectivity of neighborhoods are very important when designing representations and search operators for metaheuristics, two classes of search operators named interval-based operators (I-operators) are defined in such a way the sub-space to be explored is searched efficiently. These operators, as well as the coding/decoding tools were firstly applied to Q3AP through the algorithms HC, ILS and GA. The promising results encourage us to use those special operators in the design of hybrid methods combining tree-based exact algorithms such as B&B with metaheuristics, mainly GAs. In the next chapter, two hybridization schemes based on the concept of interval and using the representation by numbers are proposed.

Chapter 5

Tree-based hybrid optimization schemes to solve permutation problems

Contents

5.1 Introduction

Genetic algorithms (GAs) [44] have been combined with different optimization methods (heuristic or exact ones) in order to obtain a new class of methods, that explore more efficiently the search space of large and complex combinatorial optimization problems. The basic idea of such hybrids is to take advantage from the complementary behaviors of the methods they combine. In this thesis, the focus is set on combining GAs and the branch-and-bound algorithm (B&B) [65] to solve large permutation-based problems. Relevant ways to combine these two algorithms have been largely discussed in the literature and successfully applied to different classes of optimization problems ([87], [41], [21], [34]).

Those methods could be classified in two categories: relay methods working in a pipelined mode (the input of one method is the output of the previous method) or cooperative methods where the two algorithms are executed in a parallel or intertwined way and exchange information to guide their respective searches. Generally, the main information exchanged in a cooperative hybridization between GAs and B&B is the new best found solutions and the initialization of the GA's population from the B&B's pool of nodes [41], [21]. However, the advantages took from the cooperation between GAs and B&B could go beyond the traditional weakness/strength balance, best solutions exchanges, etc., if the characteristics of the two algorithms are exploited to find some "connections" between their respective search strategies.

The idea in this thesis is to reuse the B&B tree-based representation and the numbering of permutations in GAs in order to design a tight cooperating strategy between the two algorithms to efficiently explore large search spaces of permutation-based problems. In this chapter, two approaches of hybridization based on the use of the tree representation and the concept of intervals are proposed.

The first hybrid algorithm, referred to as $HGABB/HAGABB$ (Hybrid Adaptive GA-B&B), is a low-level hybridization in which the global search is managed by the GA while the B&B is used as an intensification tool embedded into the mutation operator. The second hybrid algorithm is a cooperative model in which the two algorithms GA and B&B use the same representation of the solution space in order to share effective information and help each other in their respective searches. The global search in this hybrid algorithm is managed by the B&B. In the rest of the thesis, we refer to this algorithm as $COBBIGA$ for cooperative B&B Interval-based GA.

The rest of this chapter is organized as follows: HGABB is described in Section 5.2 as well as its application to Q3AP and some results obtained for this problem. COBBIGA and the application to Q3AP are described in Section 5.3. The chapter is ended with a discussion about the re-usability of the proposed hybridization approaches for other permutation-based problems, other than Q3AP and QAP (exp. TSP, flow-shop scheduling and job-shop scheduling problem) in Section 5.5. Finally, some concluding remarks are given in Section 5.6.

5.2 HGABB/HAGABB: a low level hybrid algorithm combining GA and B&B algorithm

As population-based metaheuristics, GAs are exploration-oriented search algorithms: the use of multiple starting search points (the individuals of the population) allows a diversification of the search. On the other hand, it is acknowledged that integrating local search techniques during the different steps of a GA, enables to exploit promising solutions, as local search methods are known for their intensification capabilities. This kind of hybridization, commonly known as memetic algorithms or hybrid genetic algorithms in the literature[41] [87], allows to balance between diversification and intensification phases. Generally, the local search algorithm is encapsulated into the transformation operators and is applied either on all the population or restricted to the best individuals of the population. HGABB is based on the same hybridization strategy but rather than intensifying the search around good quality solutions using local search algorithms, the B&B algorithm is used instead, in order to search larger sub-spaces.

5.2.1 Classification of the hybrid method

According to Talbi's classification ([102]), HGABB is classified as low-level teamwork hybrid because one of the methods is used as an operator in the other method and the two methods work conjointly.

If we refer to Raidl's classification ([89]):
- What is hybridized? Genetic algorithms with tree-based exact method (B&B)
- Level of hybridization: low level, the B&B is encapsulated in a mutation operator
- Order of execution: interleaved
- Control strategy: integrative (exact techniques for searching very large neighborhoods)

5.2.2 The principle of the hybrid B&B mutation operator in HGABB

The idea in this operator (see Figure 5.1) is as follows. First, use a local search algorithm to find a good quality solution, then use the B&B solver to optimize a part of the previously found solution. If we suppose that good quality solutions are grouped in the same area, the better are the solutions found by the local search, the greater will be the probability of finding a better solution by the B&B solver.

The size of the search space to be explored by the B&B is determined by a parameter called MAX_DEPTH. This parameter determines the depth of the tree nodes to be explored by the B&B. For example, if we consider a permutation problem of size n, the first positions of the solution (permutation) until the depth MAX_DEPTH, are fixed and the remaining positions are optimally filled by the B&B. This corresponds to a tree node p of size

$$size(p) = n - MAX\_DEPTH$$

Thus, the size of the sub-space S' solved by the B&B is equal to the weight of the node p. Nevertheless, B&B is CPU time intensive. Thus, it should not be used more than once

on the same tree node. For this, a list of forbidden intervals, named I-Tabu, is carried out by the hybrid mutation operator. This list, named I-Tabu by analogy to the tabu list of moves used in the tabu search metaheuristic, contains all the explored intervals by the B&B in previous stages and is used to avoid the redundancy in the B&B's search.

Indeed, each time we get a new solution s' by the local search, the range of the node p (obtained from s') to be solved by the B&B is calculated using Equation (4.3). If $range(p) \in I - Tabu$, this means the same node had already been solved by the B&B in earlier generations of the GA. This can happen when the GA achieves convergence, or when the local search is stuck in a local optima for several generations. In that case, the solution found by the local search is the final solution of the hybrid mutation operator. Otherwise, the node p is solved by the B&B and its range $range(p)$ is added to the list of solved intervals I-Tabu. Note that this list is empty at the starting of the algorithm and unlike the tabu list in TS, it is always growing in the time and the tabu intervals remain tabu until the end of the application.

Since the elements of this list are intervals, every time a new interval is added to the list, it is either considered as a new element or merged with other adjacent intervals to form a larger interval. This helps to save memory and to support long term executions of the hybrid algorithm. Suppose that at a given time t, the list of tabu intervals is $I - Tabu = \{[a, b[\}$ and a new range $[c, d[$ is to be added to the list, then the new list $I - Tabu'$ is given by Equation (5.1).

$$I - Tabu' = [c, d[\cup \{[a, b[\} = \begin{cases} \{[min(a, c), max(b, d)[\} \;\; if \;\; d = a \;\; or \;\; b = c \\ \\ \{[a, b[, [c, d[\} \;\; otherwise \end{cases} \tag{5.1}$$

Theoretically, if the search space is entirely explored, then the list I-Tabu will contain only one element equal to the global interval which represents the global search space of the problem. If this happens, the search will be stopped and the best found solution is optimal. Nevertheless, this will probably never happen because the global search is guided by the GA and the latter is based on stochastic search operators.

5.2.3 Tunning the parameters of HGABB

A big challenge to consider here is the choice of the right parameter MAX_DEPTH for each problem size. Indeed, the larger is the size of the sub-space explored by the B&B, the greater will be the probability of finding a better solution, and the higher will be the CPU time consumed. For this issue we propose two alternatives.

- **Alternative 1:** Pilot experiments have to be done for the tackled benchmark with different node sizes to determine a balance between CPU time/sub-space size: the B&B will be used to solve nodes of different sizes, the CPU time consumed is recorded, then the right node size to be used is chosen. It is a question of balancing between size of sub-space/time by takig into account the stopping criterion.

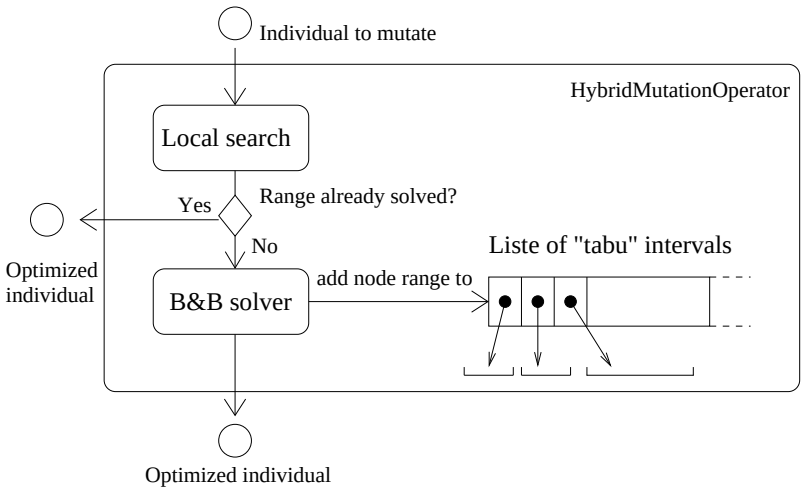


Figure 5.1: The hybrid B&B-LS mutation operator used in HGABB.

– **Alternative 2:** Use an adaptive node size which will vary through the generations and according to the solution's quality. Indeed, during the first generations of the evolution process, the individuals of the population are generally of bad quality as the search is not yet converged. Thus, the B&B will consume CPU time in exploring large sub-spaces in unpromising areas, and among all, using poor upper-bounds. This will lead to a wast of time with a small or no improvement of the best solution. The use of an adaptive node size will solve the two issues. This option is explained in details in the next section.

5.2.4 HAGABB: Adaptive Hybrid GA-B&B

In the adaptive version of HGABB, the granularity of the nodes to be solved by the B&B is determined at runtime according to the number of generations and to the solution's quality. A mechanism comparable to the cooling schedule in simulated annealing metaheuristic is used to determine the size of the tree-nodes to be explored. This mechanism has been used in some works in the literature to determine the parameter k in a k-opt operator according to the quality of the actual solution [49]. As Equation (5.2) illustrates it, the size of the node p to solve is equal to the maximum size indicated by the user if the actual solution is fitter than the best element in the population. Otherwise, the size will be calculated such that $MIN\_SIZE \leq size(p) \leq MAX\_SIZE$. With MIN_SIZE a new parameter introduced to delimit the smallest tree node to be explored by the B&B and MAX_SIZE is the size of the largest node ($MAX\_SIZE = n - MAX\_DEPTH$ with n the size of the problem).

$$size(p) = \begin{cases} MAX\_SIZE \ \ if \ \ fitness(s) > fitness(s') \\ else \\ exp(-delta/T) * (MAX\_SIZE - MIN\_SIZE) + MIN\_SIZE \end{cases} \tag{5.2}$$

$$delta = |fitness(s) - fitness(s')| \tag{5.3}$$

In Equation (5.2), *delta* is the difference between the fitness of the actual solution s and the fitness of the best element in the population s' (see Equation (5.3)). T corresponds to the temperature. Contrarily to the simulated annealing mechanism, T here is increased through generations. Indeed, when the GA approaches convergence, the quality of the best solution is very good. Thus, more and more deterioration of this fitness is tolerated. Those solutions are worth to be fine searched as they are generally of good quality as well. Thus, generally the size of the nodes to explore increases as we approach convergence. Nevertheless, the number of solutions to handle by the B&B decreases through generations as some of them have already been explored at the same size and their range is in the list I-Tabu. This version of HGABB is refereed to as HAGABB for hybrid adaptive GA-B&B algorithm.

The use of B&B to explore large neighborhoods in GA helps the latter to intensify its search around good quality solutions obtained earlier. In the next section, a second hybridization scheme between GAs and the B&B is proposed. In this new scheme the global search is guided by the B&B and the GA is used to explore new areas determined by the B&B.

5.3 COBBIGA: a new hybrid B&B/GA cooperative scheme for permutation-based problems

In a cooperative hybrid approach combining two different search algorithms, different issues expressed in the following four questions, have to be addressed: What is the information to be exchanged? What is the use made of this information in each of the cooperating algorithms? When is the communication done? and How? Moreover, in a hybrid algorithm combining a GA and a B&B algorithm, the two algorithms are heterogeneous in the sens they do not have the same objective, the same running times or the same representation of the search space. Thus, special care should be taken when the four questions listed above are answered. For example, It is mandatory to determine the information to be exchanged between the two algorithms in such a way that this information helps the global search to take maximum advantage from the characteristics of the two search methods.

In this thesis, we exploit a common problem representation in both search methods in order to get a tight cooperation. Indeed, the representation of the search space in both algorithms is based on the B&B tree and the intervals. Thus, the information exchanged is very precise because the two algorithms have the same global information about unvisited sub-spaces, promising sub-spaces, and already solved ones to escape.

The frequency of the communications is another crucial issue in the case of a cooperative scheme combining a GA and B&B. Once the useful information to be exchanged is determined, it is necessary to take into account the processing times in each method while designing the strategy of communication. The basics of the cooperative hybridization scheme B&B-GA (COBBIGA) are explained in the following sections.

5.3.1 Classification of the hybrid method

According to Talbi's classification ([102]), COBBIGA is classified as high-level teamwork method because the two methods are independent from each other and work in a parallel cooperative way to exchange information on their respective search (see Figure 5.2).

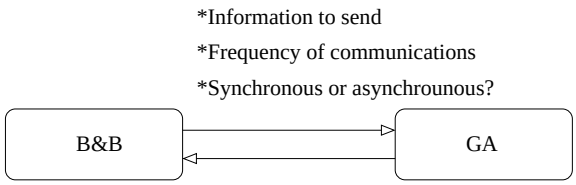


Figure 5.2: Hight level teamwork hybridization

If we refer to Raidl's classification ([89]):
– What is hybridized? Genetic algorithms with tree-based exact method (B&B).
– Level of hybridization: high level.
– Order of execution: parallel.
– Control strategy: cooperative (exact techniques for searching very large neighborhoods)

5.3.2 The strategy of cooperation in COBBIGA

As illustrated in Figure 5.3, the new hybrid B&B-GA cooperative scheme is designed as an island couple: a GA island and a B&B island. The GA island is actually using I-GA presented in the previous chapter. The two islands are able to asynchronously exchange information. The B&B island holds three pools: a pool of yet unexplored intervals (initialized with the global interval at the starting of the application), a pool of nodes to be explored (generated from the first pool) and a pool of promising nodes (empty at the beginning of the method).

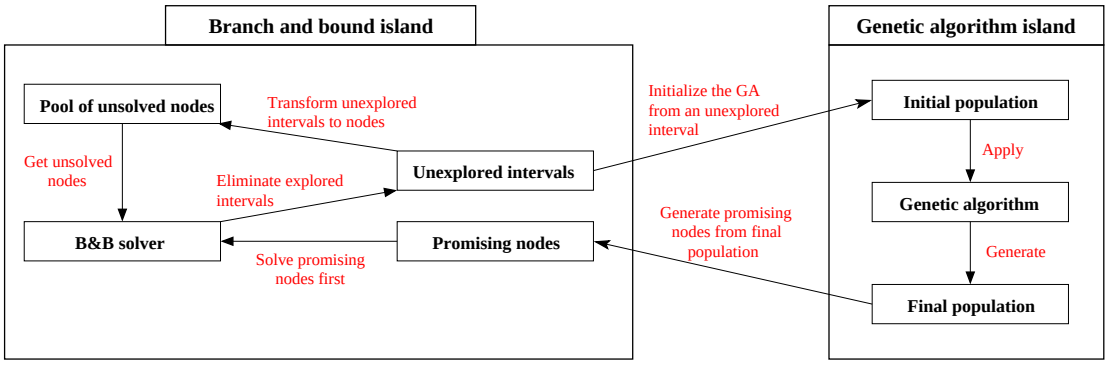


Figure 5.3: Data exchanged between the pair of B&B and GA islands in the cooperative hybrid algorithm COBBIGA.

The GA island contains a population of individuals that is initialized from the global

unexplored interval. After the initialization step, both islands start exploring in parallel their search spaces using B&B or I-GA. The B&B island starts solving sequentially the nodes in its pool. When a node is solved to optimality, its *range* (the sub-interval containing all the solutions for which the solved node is root) is deleted from the list of yet unexplored intervals.

On the other hand, I-GA evolves its population until a given number of generations MAX_GEN. This parameter also represents the migration frequency (MIG_FREQ) between the GA island and the B&B island. Afterwards, the GA island sends its final population to the B&B island in order to inform the latter of the best solution it has found and to point out promising regions.

Indeed, when the B&B island receives the population, it compares the best solution in this population to its upper bound and if it is fitter than the upper bound, the upper bound is replaced by the new best solution. Moreover, for each different solution s taken from the I-GA's population, a deviation $Delta = |fitness(s) - fitness(best)|$ is calculated. If $Delta < \omega$ this solution is accepted and is used to identify promising regions. ω is a threshold for the accepted deterioration of the best solution.

For each accepted solution s, the associated number is calculated ($number(s)$) and is used to look for an interval $[a, b[$ to whom it belongs in the list of unsolved intervals. If such an interval exists, it is used both with $number(s)$ to calculate the largest ancestor of s that belongs to the interval $[a, b[$. The ancestor is stored in the pool of promising nodes because its range may contain improving solutions to the problem. The size of the ancestor node to be processed could also be controlled by the user using a special parameter named MAX_DEPTH which is similar to the parameter of the same name used in HGABB. This procedure called *handle_received_population* is illustrated in Algorithm 14.

Afterwards, the B&B island interrupts its sequential exploration of the search space and starts solving the promising nodes first. Finally, the B&B generates a new initial population from an unexplored interval and sends it to the GA island. If the pool of intervals is empty, if the biggest interval is too small or if the time limit predefined by the user is reached, the GA island gets an empty population as a stopping criterion. The new population, sent to the GA island, represents a new sub-space to be explored and the GA's search is limited to that sub-space using I-operators (see Section 4.6).

COBBIGA is in the same time a metaheuristic and an exact method depending on the used stopping criterion, because the search is guided by the B&B. The cooperation steps between the two islands are illustrated in Figure 5.3.

5.3.3 Parameter tunning for COBBIGA

The most difficult task in this cooperative hybrid scheme is to determine the right parameters to be used in order to equilibrate the search between exploration phase done by the GA and exploitation of promising regions by the B&B. The major parameters are: the communication mode (synchronous or asynchronous), MAX_DEPTH which determines the depth of the nodes to be explored by B&B, the number of generations MAX_GEN let for

Algorithm 14: Pseudo code of the handle_received_population procedure used in COBBIGA

Input: The GA's final population P
Input: The list of yet unsolved intervals $Unsolved$
Input: Best found solution so far $best$
Output: List of promising nodes $Promising$

```
1  accept ← FALSE
2  //Main loop
3  for each solution s ∈ P do
4      if fitness(s) > fitness(best) then
5          best ← s
6          accept ← TRUE
7      else
8          Delta ← |fitness(s) − fitness(best)|
9          if Delta ≤ fitness(best) ∗ ω then
10             accept ← TRUE
11     if accept then
12         if ∃[a, b[∈ Unsolved with number(s) ∈ [a, b[ then
13             p ← ancestor(s) with range(p) ∈ [a, b[ parent(p) ∉ [a, b[
14             Add p to Promising
15 return Promising
```

the GA before it sends its final population and gets a new one, and the threshold ω which determines the acceptance criterion in the list of promising nodes for the solutions found by the GA.

Depth of the nodes (MAX_DEPTH) to be solved by B&B

This parameter also determines the size of the search space to be explored by B&B, thus it may be chosen according to (1) the objective of the search and (2) the global time available for the hybrid algorithm.

In this hybrid algorithm, the main objective is to localize global optimal solutions more quickly. Thus, the principle is to use B&B in reasonable sub-spaces determined by the GA. Indeed, if the size of the benchmark is too big for the available CPU time, it is better for B&B to solve averaged nodes located in different sub-spaces than to focus on only one large sub-space.

As in the HGABB/HAGABB algorithm presented in Section 5.2, this parameter could be determined following two ways, pilot experiments with different values or the use of adaptive size determined following Equation (5.2) (see Section 5.2). Nevertheless, in the

cooperative scheme, because the global search is controlled by B&B, it is better to use fixed node sizes in order not to create non homogeneous intervals in the list of unsolved intervals used by the B&B. An homogeneous interval is an interval which covers only tree nodes of the same size. This will simplify the work of the B&B island.

To recap, in the cooperative hybrid algorithm COBBIGA the parameter MAX_DEPTH is fixed at the starting of the application and is determined using pilot experiments. Nodes of different sizes are solved by B&B and the time required for each size is saved and the right size is chosen according to the global time limit. This point will be developed in Section 5.4. On the other hand, this setting is somehow related to the next one, MAX_GEN which determines the number of generations let to the GA island before it sends its final population to the B&B island. This parameter also represents the frequency of the communications between the two islands (MIG_FREQ). Indeed, even if the communication mode is asynchronous, the time spent by the GA until MAX_GEN generations, must be correlated to the time spent by B&B to solve one node in order to equilibrate the search between the two algorithms.

Maximum number of generations (MAX_GEN)

This setting corresponds to the number of generations let to the GA island before it sends its final population to the B&B island and gets a new one. Even if the GA's role is not limited to find the global optimum, it is necessary for I-GA (used in the GA island) to converge to good solutions in the given sub-space as this is capital for the B&B efficiency. Thus, MAX_GEN should be large enough to allow the I-GA's population to converge to the best solution(s) in the considered sub-space.

Again, this parameter is to be determined using pilot experiments. The GA's population will be recorded every generation and different metrics are calculated in order to determine the right value: time, the best fitness, average fitness of the population and the standard deviation in the population to measure the diversity. As this parameter is not sensitive to the problem instance at hand, it is not necessary to do the same experiment for each benchmark. It is enough to try on a single benchmark or even better, one benchmark for each problem size in order to measure the time required for each problem size.

Synchronous versus asynchronous communications

The problem with a synchronous communication is that the two methods do not have the same speed, even when the two parameters MAX_DEPTH for B&B and the number of generations MAX_GEN for GA are calculated in such a way that solving one node by the B&B solver takes the same time than executing MAX_GEN generations by the GA. It is impossible to find a perfect matching between the two speeds. Indeed, even if the size of the nodes to be solved by B&B is always the same, the time spent in every different node is highly variable as it depends both on the node and on the actual upper-bound in the B&B. Thus, it may happen that the B&B ends up earlier with the actual node and spends some time without working while looking forwards for the GA's last population. On the contrary, if the GA achieves the last generation but B&B is still working on the actual node, the GA stops its execution until B&B is ready to communicate. This may create a

big wast of time in the hybrid algorithm. Among all, it is not necessary for the two algorithms to synchronize their efforts, as both of them can work independently from the other.

In the asynchronous mode, when the GA achieves the last generation, it sends its last population to B&B. This population is used to create some new promising nodes that are added to the list of promising nodes, and gets a new population. On the other hand, when B&B solves one node, it takes automatically the next node to solve from the list of promising nodes. If this list is empty, it takes the next tree node to solve in a sequential order. Thus, it does not matter if the two algorithms are not progressing exactly at the same speed, both of them continue working in all cases.

Acceptance threshold ω

When the GA's final population is received by the B&B, each individual in the population is examined to decide if it is good enough to be exploited by B&B. All the individuals that are fitter than the global best found solution (the upper-bound in the B&B) are automatically accepted. The solutions for which the fitness deteriorates the upper bound are accepted if this deterioration is less than ω. In all cases, the B&B has to solve nodes until the stopping criterion (time limit or search space entirely solved). If there is no promising nodes available, the next node in the tree is solved. Generally this node can not be better (in the sens that it contains good quality solutions) than most of the nodes formed from the solutions found by the GA, especially when the size the benchmark to be solved is large. Thus, the threshold of acceptance should be large enough to allow average quality solutions to be intensified by the B&B as this is the goal of the cooperation. In the experimentations we have performed, this threshold is fixed empirically to 30% of the upper-bound.

5.3.4 The B&B Island

The B&B island is composed of three components: the memory, the B&B solver, and the communication component.

The memory is composed of the different pools: pool of unsolved intervals, pool of active nodes and pool of promising nodes to be solved first. The communication engine manages the communications with the GA island. The B&B solver is a sequential B&B algorithm used as a black box which takes a tree node as input, solves it and returns back the result. The interaction of these three components is illustrated in Figure 5.3 and already explained in Section 5.3.2. In the following, the initialization of the pool of nodes from the pool of unsolved intervals is explained.

Initialization of the B&B island

The B&B island manages two types of pools: a pool of unsolved intervals and a pool of nodes. The pool of nodes contain tree nodes whose ranges are included in one of the intervals of the pool of unsolved intervals. At the starting of the application, the pool of unsolved intervals is initialized with the global interval representing the complete search space of the tackled problem. After this initialization step, the initial interval available in

the pool of unsolved intervals is transformed into a minimal list of nodes that are stored in the second pool.

In [76], the operator used to transform an interval $[A, B[$ into a list of tree nodes is named "unfold operator". The latter generates a minimal list of tree nodes whose ranges are included in $[A, B[$, the range of their parents are not in $[A, B[$ and their union is equal to $[A, B[$ (see Equation (5.4)).

$$Unfold([A, B[) = \left\{ \begin{array}{c} node/ \\ range(node) \subseteq [A, B[\\ range(p) \nsubseteq [A, B[\\ p = parent(node) \end{array} \right. \tag{5.4}$$

In [76], this list of nodes ($Unfold([A, B[)$ is found using a B&B algorithm in which operators, except the elimination operator, are the same ones as those of B&B algorithms. At the starting of the algorithm, the root node is decomposed into sub-problems. For each sub-problem, the range is calculated. If the range is included in the interval $[A, B[$, this sub-problem is added to the list of nodes. If it is disjoint with the interval $[A, B[$, it is eliminated. Otherwise, it is decomposed to the next level and so on until the tree is completely explored.

When the interval to be transformed is large, this procedure does not take a long time. Nevertheless, when the size of the interval is small, this means we need to decompose until a certain level and to enumerate a large number of nodes in order to check if their range is included or not in $[A, B[$. Another issue is that in the case the interval $[A, B[$ is at the end of the tree, that means we are obliged to start looking for the nodes from the left to the right in the tree until the first node belonging to $[A, B[$ is reached. This unnecessary processing time could be saved if one starts looking only at the first node in the tree whose the associated number is equal to A and stops searching once the limit B is reached.

In this thesis, a new unfold operator following this principle is proposed. As illustrated in Figure 5.4, the process of the new unfold operator consists in:

- Dividing the interval into two parts in order to get the number associated to the node with the largest range included in the interval. In Figure 5.4, we get $number = 2$.

- This number is the number associated to a complete solution s in the interval. Thus, it is used to localize the largest node in the path of s for which Condition (5.5) holds. In our example, we get the node whose number is 2 at the depth 1 in the tree.

$$range(node) \subseteq [A, B[\ \ and \ \ range(parent(node)) \nsubseteq [A, B[\tag{5.5}$$

- Once the number associated to the node and its depth in the tree are known, the permutation form of the latter is generated using the decoding operator from numbers to incomplete permutations. This node is added to the pool of nodes and its range is subtracted from the interval $[A, B[$. In our example, The range of the node obtained

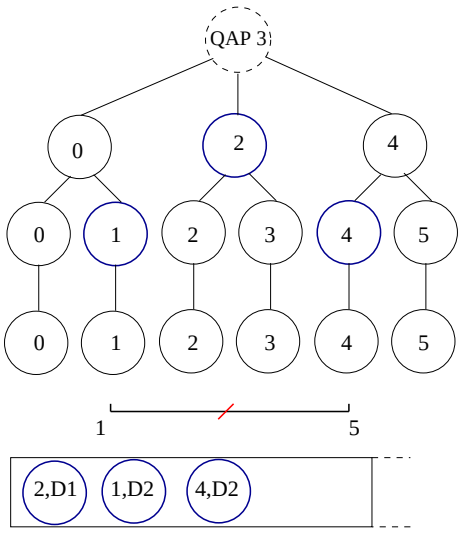


Figure 5.4: Illustration of the new Unfold operator. The interval to be transformed is $[1, 5[$ in a permutation tree for a QAP problem of size 4. The pool of nodes contains three nodes. Each node is represented by its number and its depth in the tree.

in the previous step is equal to $[2, 4[$.

– The result of this subtraction is either one or two intervals (it depends on the position of the node's range in the interval). The resulting interval(s) will be decoded recursively using the same method. In our example, we get two intervals: $[0, 2[$ and $[4, 5[$. Those intervals are decoded following the same procedure, and we get the nodes ($number = 1$, $depth = 2$) and ($number = 4$, $depth = 2$).

– The process ends if the intervals resulting from the last subtraction are empty (an interval is empty if its size is equal to 0).

The strategy of node selection in the B&B island

After the initialization step of the B&B island, two approaches are possible:

– Parallel starting: the B&B solver starts solving the nodes immediately after the initialization, and works in parallel with the GA until the stopping criterion. This approach is to be used only for the benchmarks for which we already have a best known solution to use as upper-bound in the B&B solver.

– Serial starting: the B&B solver waits for the first population of the GA in order to use the best solution in the population as upper-bound and the promising nodes provided by the latter. After this first interaction, the two algorithms work in parallel until the stopping criterion.

After this starting step, the B&B solver gets the next node to solve following two different selection strategies.

1. **Promising queue then default queue**
   Select next node from the queue of promising nodes first, if it is empty, select the next node from the default queue of unsolved nodes. Indeed, this is the main objective of the cooperation strategy between the two algorithms GA and B&B: the B&B solver should solve the promising nodes pointed out by the GA first because they are likely to contain optimal solutions. Nevertheless, in terms of time required to localize optimal solutions, this remains stochastic because it depends on the GA's stochastic search and on the time limit. For example, it might happen that the GA does not converge to the best solution in a given sub-space. Thus, this will lead B&B to solve tree nodes (from the queue of promising nodes) that are not likely to contain optimal solutions. Because the B&B's search is CPU time intensive, this situation will lead to a larger time to localize the global optimum. In the worst case, for some benchmarks, if the optimal solution is located at the beginning of the search tree, the results obtained by the hybrid approach could be worst than those of the B&B alone (in terms of time to reach optimum).

2. **Best evaluation from the two queues**
   In order to address the issue described above, a second criterion can be added to the selection strategy. An evaluation function can be used to estimate the quality of the next node in the default queue. This quality is compared to that of the next node in the queue of promising nodes. Then, the best node is explored first by the B&B solver. With this new strategy, in the worst case, the hybrid algorithm will perform equally to the B&B alone. Ideally, the evaluation function is a lower bound function for the tackled optimization problem. Indeed, in many works dealing with the B&B algorithm, the nodes are solved according to the quality of their lower bound [76]. Each time a tree node is decomposed, the bounds of all the generated sub-problems are calculated and those nodes are inserted in the queue following the quality of their associated bound. The nodes with a good lower bound are solved first. Nevertheless, for some problems, this function is CPU time intensive, thus it could be replaced by an evaluation function which simulates the decomposition of the actual node in order to estimate the quality of the sub-problems in the next level of the tree.

The B&B solver

The B&B solver consists of a sequential B&B algorithm which takes a single tree node as input, and returns the result to the island. Indeed, the solver is used as a black box by the B&B island in order to separate the different components and facilitate the adaptation of the hybrid scheme to different permutation-based problems.

5.3.5 The GA island

Initialization

Initial population for the GA is always taken from the pool of yet unsolved intervals hold by the B&B island. This pool is updated by the B&B solver each time a tree node is solved. Thus, the sub-spaces that are already solved by B&B are excluded from the GA's search. At the starting of the application, the GA is initialized from the global interval representing the entire search space of the tackled problem, which is the only interval available in the B&B's pool. The interval-based initialization method is used to generate a population from a given interval (see Section 4.5).

Transformation operators

As the GA is supposed to search in a given sub-space determined by an interval, the transformation operators as well as the initialization, are interval-based. As already said, the objective of the GA is not to locate the global optimum but to locate the best solutions situated in the considered sub-space in order to be explored later by B&B. For example, the crossover I-PMX and the combined mutation (I-ILS/I-shift) presented in the previous chapter of this thesis, could be used here.

5.3.6 Expected impact of the hybridization on the global search

In COBBIGA, the B&B algorithm is guiding the global search while the I-GA is used by the latter to explore new regions and to localize good quality solutions. Those solutions will be latter used to point out the B&B algorithm to promising tree nodes to be solved first. Thus, the impact of the cooperation on the B&B's search consists of two factors. First, decreasing the time required to find good quality solutions, hopefully optimal solutions. Second, the fast improvement of the upper bound helps to reduce the time required to solve one tree node as well as the global search of the B&B solver. Indeed, in the B&B algorithm the better is the upper-bound, the faster is the time required.

Alternatively, the GA takes advantage from the cooperation to diversify its search and at the same time, to restrict it to sub-spaces that are not optimally solved by B&B. The GA's population is regularly renewed, after a given number of generations (MAX_GEN), from the pool of unsolved intervals hold by the B&B.

Globally, the use of a common representation of the search space (the tree and the intervals) in both algorithms makes the cooperation very tight in the sens that both algorithms are aware about the sub-spaces that are visited by each other. This reduces the redundancy in the global search and diversifies it to unsolved sub-spaces. This interminable dynamic makes this hybrid method very robust as a metaheuristic. Indeed, finding optimal solutions is only a question of time, there is no premature convergence problem and it is impossible for the search to be stuck in a local optima because it is always progressing as it is guided by the B&B.

5.4 Application to Q3AP

In order to evaluate the performances of the two sequential hybrid algorithms proposed in this chapter, an implementation based on the two frameworks ParadisEO [17] and BOB++ [27] is proposed for Q3AP. In the following, the two algorithms applied to Q3AP are detailed as well as the results obtained on standard Q3AP benchmarks.

5.4.1 Implementation of HGABB/HAGABB for Q3AP

In this section we will give the implementation details of HGABB/HAGABB for Q3AP.

5.4.1.1 The B&B algorithm for Q3AP

The B&B algorithm used in the experiments on Q3AP benchmarks is developed in the BOB++ framework [27] and is kindly provided by Galea and Le Cun [40]. This algorithm uses a bound function based on a dual procedure that solves the level-1 RLT (linearization reformulation technique) formulation of the Q3AP. This lower bound proposed by Hahn *et al.* in [47], is an extension of the QAP lower bound by the same authors. We used the sequential version of the algorithm.

5.4.1.2 The GA for Q3AP

The GA used in the experiments for Q3AP is implemented using the ParadisEO framework [17]. The main components of the algorithm (initialization, selection, crossover and mutation) are described in the following.

Initialization could be either random or interval-based or any other initialization method. However, since intervals will be used during the mutation phase in order to check if a given node has already been solved by the B&B solver or not, it is worthy to initialize the population using the interval-based initialization method presented in Section 4.5 in order to have nodes from different sub-trees in the B&B tree. However, the search of the GA is global. The selection used is Roulette Wheel selection with elitism. The crossover used is I-PMX applied to the two permutations of the Q3AP solution. The mutation is a combination between I-shift mutation (applied to the two permutations of the Q3AP solution) and a hybrid mutation operator which first apply a local search algorithm on the individual to mutate followed by the BOB++ B&B solver in order to intensify the search.

Algorithm 15 is an implementation of the hybrid mutation operator used in the two versions of the algorithm HGABB and HAGABB. The inputs of the algorithm are the individual to mutate, the temperature, the list of tabu intervals (intervals already solved by B&B in previous generations) and the two parameters determining the upper-bound and lower-bound of the size of nodes to be solved by B&B (MAX_SIZE and MIN_SIZE). This algorithm is used for the two versions of the hybrid GABB algorithm. In the standard version, the size of the nodes to be solved is fixed and is obtained simply by setting MAX_SIZE equal to MIN_SIZE.

Algorithm 15: Pseudo code of the hybrid mutation operator combining a local search and a B&B algorithm, used in the two versions of the hybrid GABB algorithm (HGABB and HAGABB)

Input: Individual or solution to mutate s, f the fitness of s
Input: The list of already solved intervals: $I - Tabu$
Input: The temperature T
Input: MAX_SIZE and MIN_SIZE two parameters to determine the granularity of the nodes to be solved by B&B

```
1  s' ← apply_local_search(s)
2  f' ← fitness(s')
3  delta ← |f' − f|
4  if f' > f then
5  |   s ← s'
6  |   f ← f'
7  |   NODE_SIZE ← MAX_SIZE
8  else
9  |   NODE_SIZE ← e^(−delta/T) · (MAX_SIZE − MIN_SIZE) + MIN_SIZE
10 TemperatureUpdate(T)
11 NODE_DEPTH = N − NODE_SIZE
12 NODE ← generate-q3ap-node(s, NODE_DEPTH)
13 if range(NODE) ⊈ I − Tabu then
14 |   set-BB-upperBound(f)
15 |   s' ← applyBB(NODE)
16 |   if f' > f then
17 |   |   s ← s'
18 |   |   f ← f'
19 return s
```

In the adaptive algorithm (HAGABB) the size of the nodes to be solved is calculated according to the parameters: temperature, MAX_SIZE, MIN_SIZE and the difference between the new solution obtained by the local search and the fitness of the initial solution. Note that if the solution obtained by the local search is fitter than the actual solution (the individual to mutate), the size of the node to be solved is fixed to MAX_SIZE in order to intensify the search around the new solution. The temperature is updated afterwards. It is actually increased by 0.5 after each application of the hybrid mutation, in order to increase the granularity of the nodes to be solved by B&B as the GA is approaching to convergence.

Once the size of the node to be solved is known, its depth in the tree is deduced and used to generate a Q3AP node from the actual solution. The Q3AP node is a copy of the Q3AP solution in which the positions from 0 to $NODE\_DEPTH$ in the two permutations are fixed,

and the remaining are to be filled optimally by the B&B solver. The number and the range of the obtained node are calculated in order to check if the range is not already solved by the B&B solver which means the range is not included in the list $I - Tabu$. If it is not, the node is solved otherwise it is ignored. The B&B upper-bound is initialized to the fitness of the solution obtained by the local search instead of the best solution in the population, because the search of the B&B is not supposed to be global. Indeed, the role of the B&B solver here is not to find a global solution to the problem but to improve the solution obtained by the local search step. Conversely, the global search is guided by the GA.

5.4.1.3 General settings

Pop.Size	**Cross.Rate**	**Mut.Rate**	**ILS.ITER.**
60	1	0.4	5
MAX_SIZE	**MIN_SIZE**	**T**	**MAX_GEN**
9	6	100	600

Table 5.1: Global settings used by HGABB/HAGABB

The parameters used in the experiments are shown in Table 5.1. The two parameters MAX_SIZE and MIN_SIZE are chosen in such a way to balance between the size of the search space explored by B&B and the required CPU time. The initial temperature T is chosen empirically.

5.4.1.4 Experimental results

In order to evaluate the performances of the two hybrid algorithms HGABB and HAGABB experiments are carried out on the following Q3AP benchmarks: Nug12, Nug12c, Nug13, Had14 and Nug15. Three algorithms are considered for comparison: HGABB, its adaptive version HAGABB, and GAILS (a GA in which the mutation is a combination between a shift operator and the ILS algorithm). The last algorithm is used to show the added value of the use a B&B solver on the best solution found by the local search step in HGABB/HAGABB. Each algorithm is executed 30 times on each benchmark. The machine used in these experiments is a 64 bits 2.6 GH machine.

The variation of the granularity of nodes to be solved by the B&B solver in HAGABB is illustrated in an example of execution on the benchmark Nug12 (see Figure 5.5). In Figure 5.5, it is noticed that at the starting of the algorithm, the nodes to solve have the smallest size and the size is growing through generations to reach the size limit given by the parameter MAX_SIZE. The size is influenced both by the temperature and the quality of the solutions.

The average results achieved by the three algorithms after 600 generations are shown in Table 5.2. The column "Hits" represents the percentage of the runs in which each algorithm finds the best known solution of the benchmark over the 30 runs. The columns A.Fit, SD and A.Time refer to the average fitness values, fitness standard deviation and average execution time in seconds respectively. Note that the time measured is the real time of

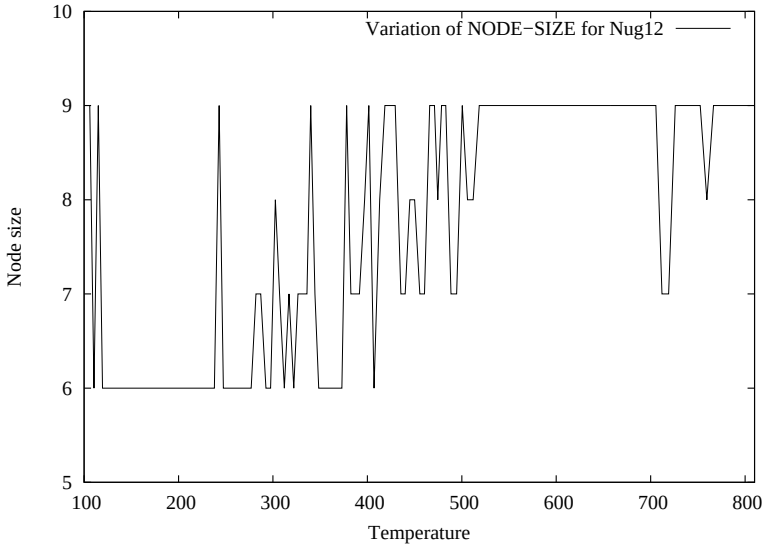


Figure 5.5: Variation of $NODE\_SIZE$ in HAGABB on the benchmark Nug12.

execution on a dedicated machine.

The results are analyzed in the sequel. The average fitness values through generations achieved by the three algorithms on a set of Q3AP benchmarks of different sizes (Nug12, Nug12c, Nug13, Nug15 and Had14) are shown in the Figures 5.6 (a), 5.6 (b), 5.7 (a), 5.7 (b) and 5.8 respectively. The three algorithms perform equally on Nug12 while HAGABB outperforms HGABB and GAILS in the benchmarks Nug13, and Nug15, HGABB outperforms HAGABB and GAILS on the benchmarks Had14, and GAILS outperforms the two algorithms HGABB and HAGABB on the benchmark Nug12c. None of the algorithms is clearly better than the two other algorithms on all the benchmarks.

In terms of execution times required by each algorithm, the GAILS is faster in all cases as it does not contain a B&B step. On the other hand, the adaptive strategy used in the HAGABB to determine the granularity of the nodes to be solved clearly helps to save time compared to the fixed sizes in HGABB. Figure 5.9 shows the average execution times required by each one of the three algorithms on all the benchmarks through generations.

To conclude, generally HGABB/HAGABB performs slightly better than GAILS in terms of solution quality but not on all the benchmarks and requires more CPU time. However, we are convinced that a better usage of the B&B solver will be made if the global search is guided by the B&B using the B&B tree. Indeed, since the global search in the HGABB/HAGABB algorithm is guided by the GA, the search remains stochastic and there is no guarantee to find the global optima no matter the execution time allocated to the algorithm, even if a B&B solver is used to intensify the search around promising solutions.

Bench.	GAILS				HGABB				HAGABB			
	Hits(%)	A.Fit.	SD	A.Time(s)	Hits(%)	A.Fit.	SD	Time(s)	Hits(%)	A.Fit.	SD	Time(s)
Nug12	90	590	34	4012	92	584	23	5005	93	589	35	4673
Nug12c	51	1337	14	4036	41	1341	16	5855	37	1341	18	4446
Nug13	8	1978	41	5388	6	1979	38	5851	7	1966	38	5354
Had14	97	37602	22	5029	100	37598	0	10067	100	37598	0	7829
Nug15	32	2414	132	5461	48	2390	155	8176	65	2338	143	7586

Table 5.2: Average results obtained by the algorithms GAILS, HGABB and HAGABB on the Q3AP benchmarks Nug12, Nug12c, Nug13, Had14 and Nug15.

Conversely, in the cooperative hybrid scheme COBBIGA, this limitation is bypassed because the global search is managed by B&B. The B&B tree is used to drive the GA exclusively towards unsolved sub-spaces, while the B&B solver is used to solve promising sub-spaces pointed out by the GA. The dynamic of the cooperative search and the use of the B&B tree to memorize the historic of the global search, makes this new hybrid algorithm less stochastic and the global optimum will be found sooner or later. The objective of the next set of experiments (Section 5.4.2) is to check the effectiveness of this assertion.

5.4.2 Implementation of COBBIGA for Q3AP

5.4.2.1 The GA island

The implementation of the GA island in COBBIGA is illustrated in Algorithm 16. The first population is initialized from the global interval representing the global search space of the tackled benchmark. After initialization, the I-GA is executed (a GA in which the transformation operators are interval-based). When the number of generations reaches the parameter MAX_GEN, which determines the migration frequency to the B&B island, the final population of the GA island is sent to the B&B island. This population will be used by B&B to create new promising nodes to be explored and a new population is generated from one of the remaining intervals in the B&B pool (representing a new unexplored sub-space). If there are no more intervals in the B&B pool or the time limit is reached, the GA island will receive an empty population which means the search is ended.

Algorithm 16: Implementation of the GA island in COBBIGA.

```
1  i ← 0
2  Initialization step
3  Initialize the first population P from the global interval I = [0, n!²[
4  Evaluate(P)
5  while P is not empty do
6      P' ⇐ RouletteWheelSelection(P)
7      P'' ⇐ Interval − based − genetic − operators(P')
8      Evaluate (P'')
9      P ⇐ Replacement(P'', P)
10     if Migration frequency reached then
11         Communicate P to the B&B island
```

5.4.2.2 The B&B island

The mechanisms of initialization of the B&B island and the cooperation with the GA island have already been explained in the previous sections. Like in HGABB/HAGABB, the B&B solver is used in COBBIGA as a black box in order to make the hybrid algorithm easily reusable for other permutation-based problems. This black box takes a tree node and an upper-bound as entrance and gives back a Q3AP solution if the initial solution has been

improved. Here the B&B solver is initialized with the global best found solution because the objective of the search is to find the optimal solution as soon as possible. The Q3AP B&B solver has already been described in Section 5.4.1.1.

5.4.2.3 Experimental results

As already pointed out, COBBIGA is considered at the same time as an exact algorithm if it is executed until the search space is completely explored and as a metaheuristic if any other stopping criterion is used. Thus, the objective of the experiments reported in this section is to show, on one hand, the impact of the cooperation between B&B and I-GA on the global search guided by the B&B compared to a B&B algorithm without cooperation. On the other hand, if COBBIGA is considered as a metaheuristic aiming to find good quality solutions, a comparison is done with a standard GA algorithm enhanced with local search intensification, more exactly, the GAILS algorithm used in the previous section.

Parameter tuning in COBBIGA

The parameters used in the I-GA are listed in Table 5.3. Population size, crossover rate and mutation rate as well as the number of iterations of the local search (the I-ILS used as mutation operator) are fixed as they do not influence the cooperation mechanism. Alternatively, in order to determine optimal values for the two parameters MAX_DEPTH and MAX_GEN which determine respectively, the depth (and the size) of the nodes to be solved by B&B and the number of generations let for I-GA before it sends its final population and it receives a new one. Two values are considered for each parameter. As these two parameters are slightly related, they are used in pairs: {1, 80} and {2,50}.

These couples of values have been chosen empirically according to the results of pilot experiments in which the average execution time required by the B&B to solve tree nodes of different depths in the tree and the average time required by the I-GA to execute one generation, both on a Q3AP benchmark of size 12, are calculated, as well as the average number of generations needed by the I-GA to reach convergence to local optima in the given interval. The two parameters MAX_DEPTH and MAX_GEN are then chosen in a way to have an equivalence between the execution time of the B&B on one tree node and the execution time required by I-GA, in order to balance the search effort between the two algorithms.

Pop.Size	**Cross.Rate**	**Mut.Rate**	**I-ILS.ITER.**
60	1	0.4	2
MAX_DEPTH	**MAX_GEN**	**Nb.Migrants**	**Stop.criterion**
{1,2}	{50,80}	60	empty pop.received

Table 5.3: Global settings used for I-GA in COBBIGA.

The results obtained by COBBIGA using the two couples of values for the parameters MAX_DEPTH and MAX_GEN ({1, 80} and {2,50}) are shown in Table 5.4 in

which COBBIGA-1 and COBBIGA-2 refer to the couples of values $\{MAX\_DEPTH = 1, MAX\_GEN = 80\}$ and $\{MAX\_DEPTH = 2, MAX\_GEN = 50\}$ respectively. The average time took by each configuration to reach the global optimum of each benchmark is given in the third column. It is clearly noticed that the second configuration $\{MAX\_DEPTH = 2, MIG\_SIZE = 50\}$ gives better results for all the considered benchmarks of size 12.

Bench.	Algo.	Average time (s)
Had12	COBBIGA-1	8944
	COBBIGA-2	5301
Had12b	COBBIGA-1	1934
	COBBIGA-2	320
Nug12	COBBIGA-1	4411
	COBBIGA-2	2249
Nug12c	COBBIGA-1	11354
	COBBIGA-2	3570

Table 5.4: Results obtained by COBBIGA using the parameters $MAX\_DEPTH = \{1, 2\}$, $MAX\_GEN = \{50, 80\}$ on the Q3AP benchmarks Nug12, Nug12c, Had12 and Had12b.

Figures 5.10, 5.11, 5.12 and 5.13 illustrate the average fitness values through the time (in seconds) obtained by the two configurations on the Q3AP benchmarks Had12, Had12b, Nug12 and Nug12c, respectively.

The next set of experiments consists in executing the algorithms B&B and GAILS until the global optimum is reached and comparing those results to the results obtained by COBBIGA with the configuration $\{MAX\_DEPTH = 2, MAX\_GEN = 50\}$.

Evaluation of the performances of COBBIGA against B&B alone

Because the global search is managed by the B&B in COBBIGA, these experiments are CPU intensive compared to HGABB/HAGABB. Thus, the Q3AP benchmarks used in the following experiments are of moderate size (12): Nug12, Nug12c, Had12 and Had12b. Larger benchmarks are tackled using the parallel version of COBBIGA (see next chapter). All the experiments in this section are performed on a quad-core 64 bits, 2.6 GH machine.

Since the B&B algorithm alone is deterministic, only one run is needed for this algorithm. On the other hand, in COBBIGA the global behavior of the hybrid algorithm is stochastic regarding the execution times required to reach optimum because the efficiency of the B&B solver to localize global optima depends on the GA's search. Thus, we need several runs for COBBIGA. The reported results are the average values among 10 independent runs.

The results are summarized in Table 5.5. The third column gives the average time

required by each algorithm to reach the optimum for each benchmark. In all the considered benchmarks, COBBIGA outperforms B&B. For the benchmarks Had12 and Had12b COBBIGA is 11 times faster than B&B alone. In the case of Nug12 COBBIGA is 5 times faster and 35 times faster in the case of Nug12c.

When compared to the B&B algorithm alone, COBBIGA explores less nodes to reach optimum. Thus, it also requires less time in average. The time saved is also due to the early improvement of the upper-bound in the B&B solver thanks to the cooperation with I-GA.

Bench.	**Algo.**	**Average time (s)**
Had12	COBBIGA	5301
	B&B	61749
Had12b	COBBIGA	320
	B&B	3604
Nug12	COBBIGA	2249
	B&B	12961
Nug12c	COBBIGA	3570
	B&B	127425

Table 5.5: Results obtained by COBBIGA using the parameters $MAX\_DEPTH = 2$, $MAX\_GEN = 50$ and the B&B algorithm alone with MAX_DEPTH=2 on Q3AP benchmarks of average size (Nug12, Nug12c, Had12 and Had12b).

In order to further understand the role of I-GA in COBBIGA, a case study is done on one run of the two algorithms COBBIGA and B&B on the benchmark Had12. The log files generated by the B&B algorithm and COBBIGA are analyzed in Figure 5.14 and Figure 5.15 respectively.

The global search space of this benchmark is represented by a rectangle. The length of the rectangle corresponds to the size of the search space: $12!^2$. In Figure 5.14, the B&B algorithm solves the nodes in a numerical order (from $num = 0$ to $num = 12!^2$). For each colored surface which represents an explored sub-interval by the B&B solver, the instant time (T) at which the sub-interval is solved, is displayed. The final solution is also given (no improvement in the case of Figure 5.14). In Figure 5.15, COBBIGA solves the nodes in the order determined by the I-GA's final populations. Thus, the nodes containing good quality solutions are solved first. This leads the B&B to find the optimal solution faster.

It is important to emphasis that in COBBIGA, the best solutions are generally found by the B&B solver when exploring the promising regions pointed out by I-GA. This proves the efficiency of the cooperation scheme in localizing high quality solutions.

Evaluation of the performances of COBBIGA as a metaheuristic

In this section, COBBIGA is considered as a metaheuristic and is compared to a GA without cooperation with a B&B. The algorithm used in the comparison is again GAILS. 10 runs are performed for each algorithm on the benchmarks Nug12, Nug12c, Had12 and Had12b. The development over time of the average fitness achieved by each algorithm on the four benchmarks is shown in Figures 5.18, 5.19, 5.16 and 5.17.

The two algorithms perform equally for the benchmarks Had12, Had12b and Nug12. Both of them succeed in finding the optimal solution in 100% of the runs after 2 hours in average. For Nug12c, only COBBIGA obtains 100% of successful runs after 3 hours run.

Some benchmarks are easier than others even if they are of the same size. As a metaheuristic, COBBIGA has the advantage of using the B&B tree to continuously diversify the search to unexplored sub-spaces, avoid redundancy by exploring only the sub-spaces which are not yet visited by the B&B solver and efficiently intensify the search around good quality solutions found in each sub-interval. This dynamic of the search increases the probability of convergence of COBBIGA towards global optima. Indeed, finding the global optimum with COBBIGA is only a question of time, later or sooner the algorithm will reach the optimum. Alternatively, in a pure metaheuristic such as GAILS, the search is blind there is no clear search guidance in the search space and the algorithm could be stuck in a local optima if no continuous diversification mechanism is used. This difference will be more clear when the tackled benchmarks are larger.

5.5 Re-usability for other permutation-based problems

The two hybrid algorithms HGABB/HAGABB and COBBIGA are intended to be used for any permutation-based problem. In this chapter, the problem used for application is Q3AP. Since neighborhood and genetic operators applied to Q3AP could necessarily be applied to QAP. The implementation of the two algorithms proposed for Q3AP could be reused for QAP with few modifications related to the representation of a candidate solution (one permutation instead of two), evaluation of a solution and a B&B solver for QAP. The skeleton of the two hybrid algorithms will remain unchanged regardless of the application problem. Same Remark for any other assignment problem and absolute position permutation problems generally.

For other permutation problems in the two classes relative order and absolute order problems (such as TSP and Flowshop), it is necessary to implement interval-based versions for the relative order neighborhoods and genetic operators such as insertion and relative order crossover (OX ans UOX). This task is not too different compared to the implementation of the I-2opt neighborhood and I-PMX. All the reviewed operators for permutation-based problems in Chapter 3 are based on small modifications performed in a selected point or segment of the permutation. It does not matter which kind of modifications: swapping of two positions, insertion of one element or a segment of elements in a chosen position

of the permutation, reordering a segment of the permutation, etc. It is always possible to design either an equivalent permutation-based I-operator or tree-based one by determining a minimum cutting point i in the permutation let it be π, and all the required basic operations of the considered operator are performed in the segment: $< \pi(i+1), \pi(i+2), \ldots, \pi(n) >$.

Once the basic interval-based operators are adapted for this class of permutation problems, HGABB/HAGABB and COBBIGA could be reused for TSP and any other problem of the same class. Same remark for absolute-order permutation problems.

Nevertheless, there is a unique restriction: it applies only for problems using direct permutation representations. For example, in the case of JSP (Jobshop Scheduling) an indirect permutation representation is used and problem-specific neighborhoods and genetic operators are required [10]. Indeed, in this problem a permutation with repetition is used and the exact size of the search space can not be represented by intervals because each solution of the problem needs a decoding step before it can be optimized. Thus, the hybridization strategies proposed in this thesis do not apply to JSP.

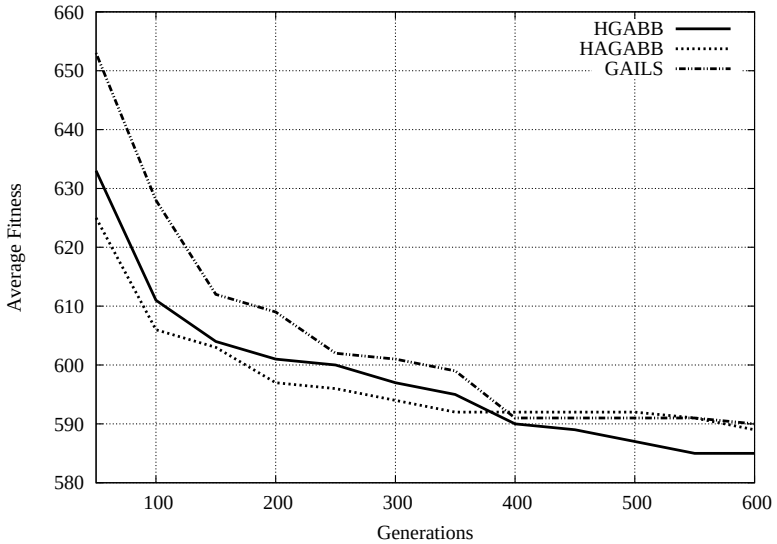
5.6 Conclusion

Hybrid methods combining exact search algorithms with metaheuristics have been largely studied in the literature in the last decade for different types of problems. In this thesis, the focus is set on efficient hybridization strategies between tree-based exact algorithms such as B&B, and GAs. These two classes of methods have divergent search objectives and strategies. GAs are based on stochastic evolution operators and other mechanisms inspired from the theory of evolution, while B&B algorithms are based on an implicit enumeration of the solution space which is represented as a tree.

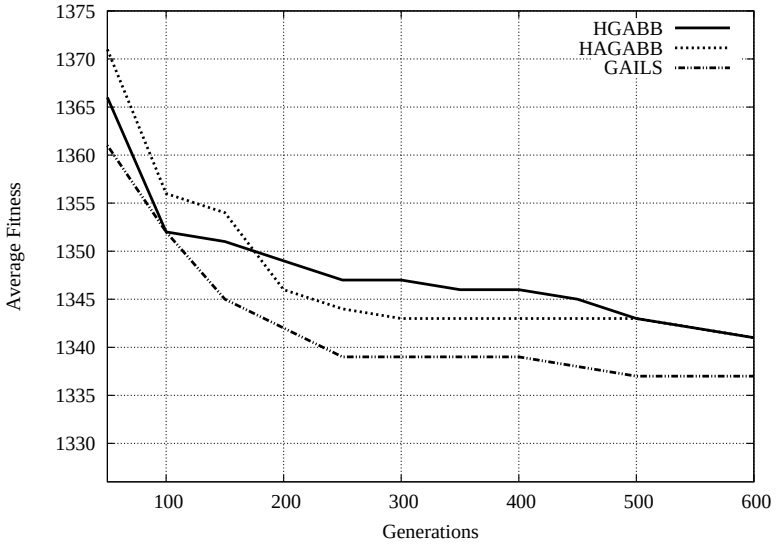
The contribution highlighted in this chapter consists in using a common representation of the solution space in the two algorithms B&B and GA, in order to design efficient loosely coupled low-level and high-level hybridizations strategies for permutation-based problems. Indeed, the coding functions proposed in the previous chapter to map between the space of permutations and natural numbers representing their lexicographic order, are used in this chapter to find some connections between the divergent search strategies used in the two classes of methods. Two hybridization schemes combining GAs with B&B, based on this common representation are proposed. The Hybrid GA-B&B algorithm (HGABB) and its adaptive version (HAGABB) are low-level hybridizations in which a B&B solver is embedded in a mutation operator in order to intensively explore large sub-spaces. A memory module similar to the tabu list used in tabu search algorithms is used here in order to avoid redundant work in the B&B solver. Indeed, a list of already explored intervals is maintained inside the mutation operator and updated every time the B&B solves a given sub-space to optimality. The second hybrid algorithm, called COBBIGA, is a high level cooperative method in which the two algorithms B&B and I-GA exploit a common representation of the solution space and asynchronously exchange best found solutions, sub-spaces to intensify and information about already explored regions to avoid. The B&B tree and the concept of interval are used in the B&B's side while in the GA's side. The interval-based search

operators presented in the previous chapter are used to limit the search of the I-GA to the sub-spaces determined by the B&B.

Parameter tuning in COBBIGA is done empirically in order to determine optimal values for the most important parameters: the size of the sub-spaces to be solved by B&B and the frequency of communications between the two algorithms. The two hybrid schemes COBBIGA and HGABB/HAGABB have been implemented and their performances are compared to the performances of GAs and B&B used separately, when solving standard benchmarks of Q3AP. The preliminary results show the effectiveness of the two approaches. Nevertheless, the use of B&B on large benchmarks leads to expensive CPU times. Thus, in the next chapter, a parallelization of the two hybrid schemes is proposed in order to solve larger benchmarks of Q3AP over a computational grid.

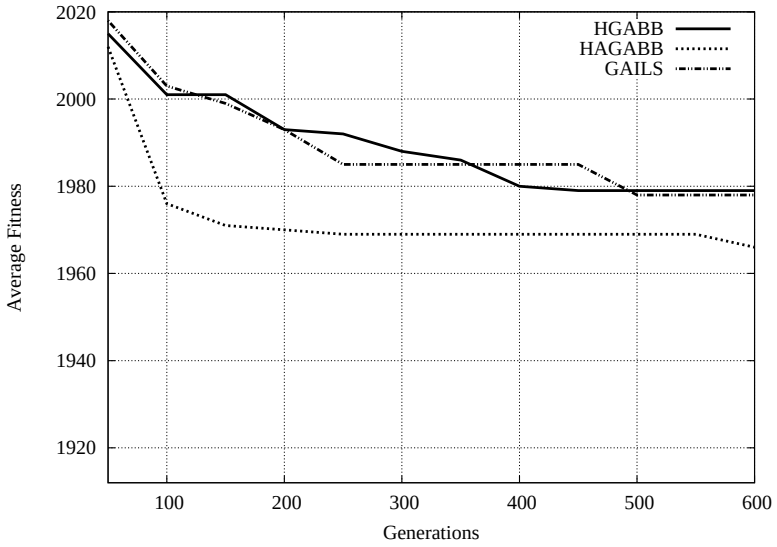


(a) Average fitness values over 30 runs obtained by the algorithms GAILS, HAGABB and HGABB on the Q3AP benchmark Nug12.

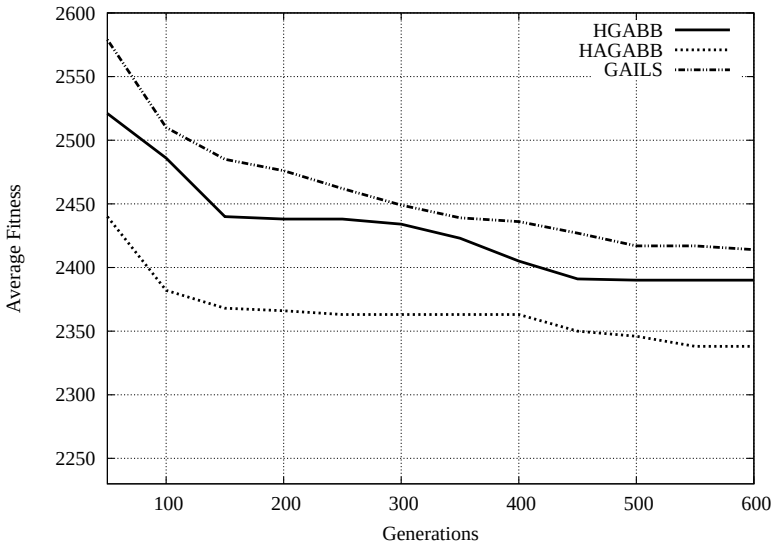


(b) Average fitness values over 30 runs obtained by the algorithms GAILS, HAGABB and HGABB on the Q3AP benchmark Nug12c.

Figure 5.6: Statistical results obtained using HAGABB, HGABB and GAILS on Nug12 and Nug12c.



(a) Average fitness values over 30 runs obtained by the algorithms GAILS, HAGABB and HGABB on the Q3AP benchmark Nug13.



(b) Average fitness values over 30 runs obtained by the algorithms GAILS, HAGABB and HGABB on the Q3AP benchmark Nug15.

Figure 5.7: Statistical results obtained using HAGABB, HGABB and GAILS on Nug13 and Nug15.

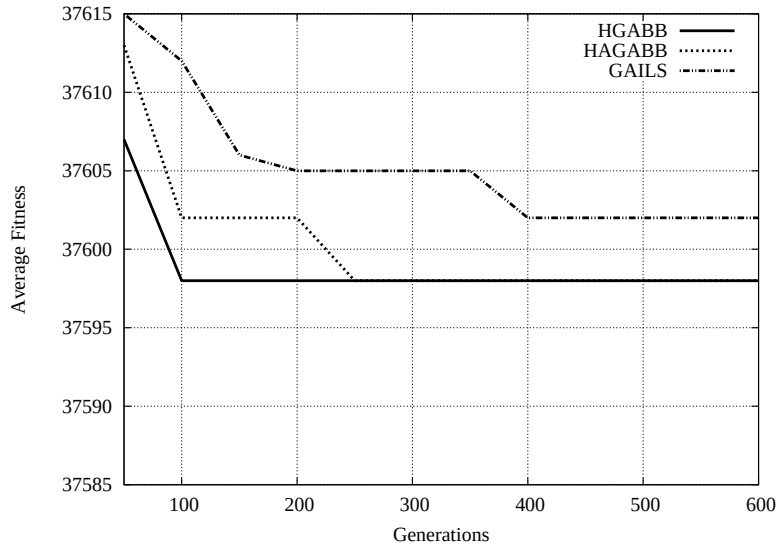


Figure 5.8: Average fitness values over 30 runs obtained by the algorithms GAILS, HAGABB and HGABB on the Q3AP benchmark Had14.

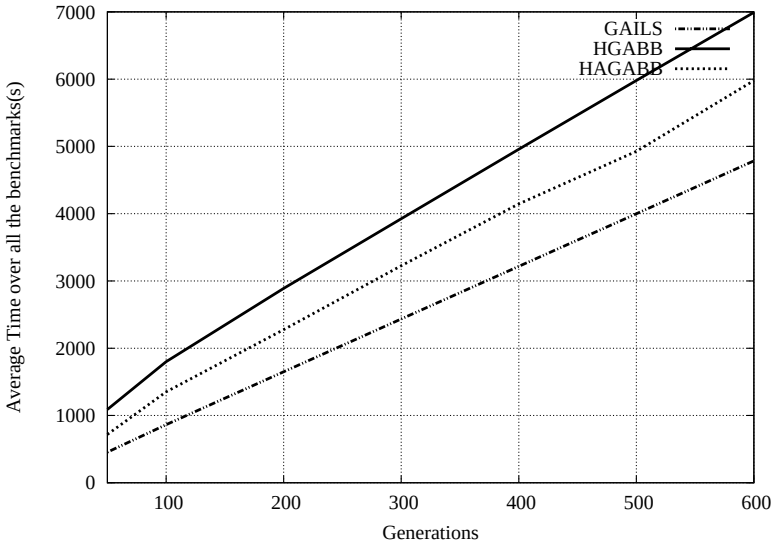


Figure 5.9: Average execution times (s) required by the algorithms GAILS, HAGABB and HGABB on all the benchmarks.

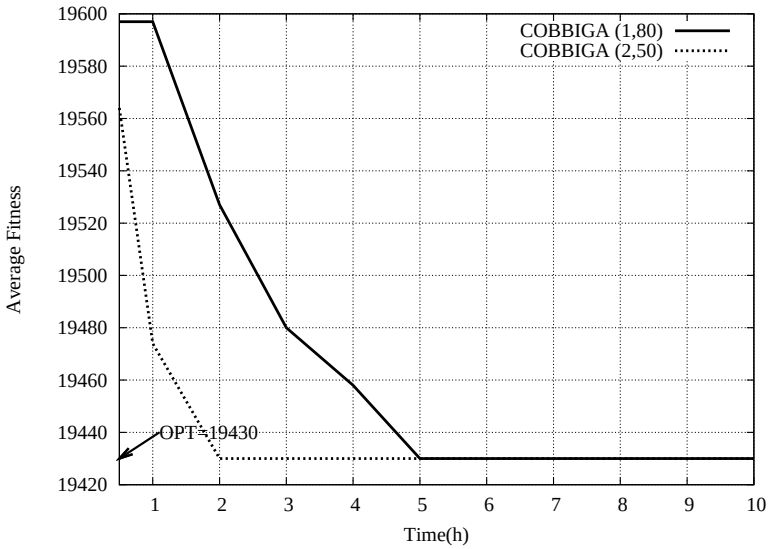


Figure 5.10: Average fitness values obtained by COBBIGA using the parameters MAX_DEPTH={1,2}, MAX_GEN={80,50} on the benchmark Had12.

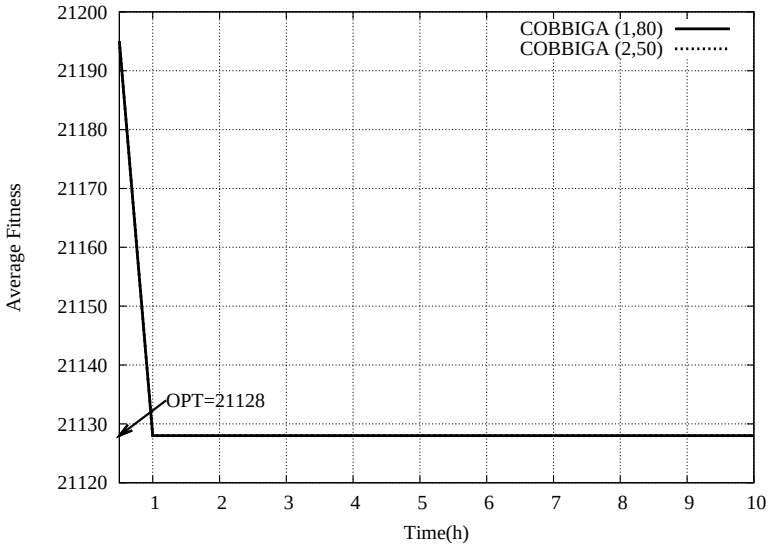


Figure 5.11: Average fitness values obtained by COBBIGA using the parameters MAX_DEPTH={1,2}, MAX_GEN={80,50} on the benchmark Had12b.

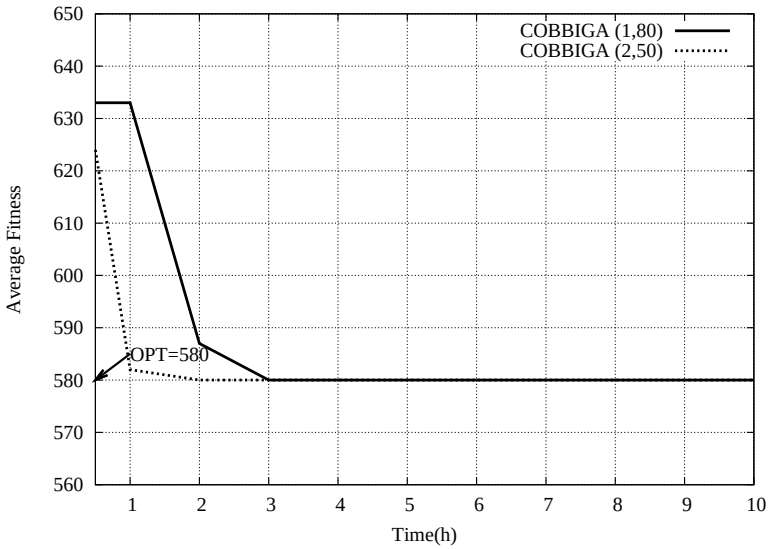


Figure 5.12: Average fitness values obtained by COBBIGA using the parameters MAX_DEPTH={1,2}, MAX_GEN={80,50} on the benchmark Nug12.

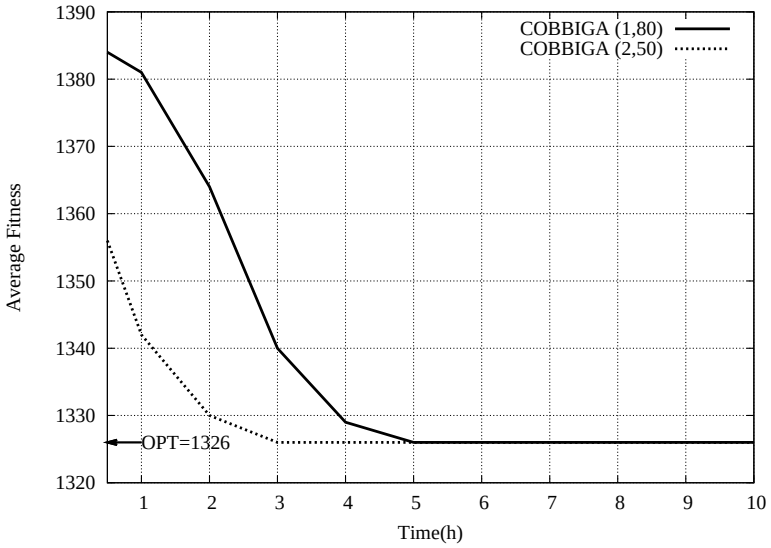


Figure 5.13: Average fitness values obtained by COBBIGA using the parameters MAX_DEPTH={1,2}, MAX_GEN={80,50} on the benchmark Nug12c

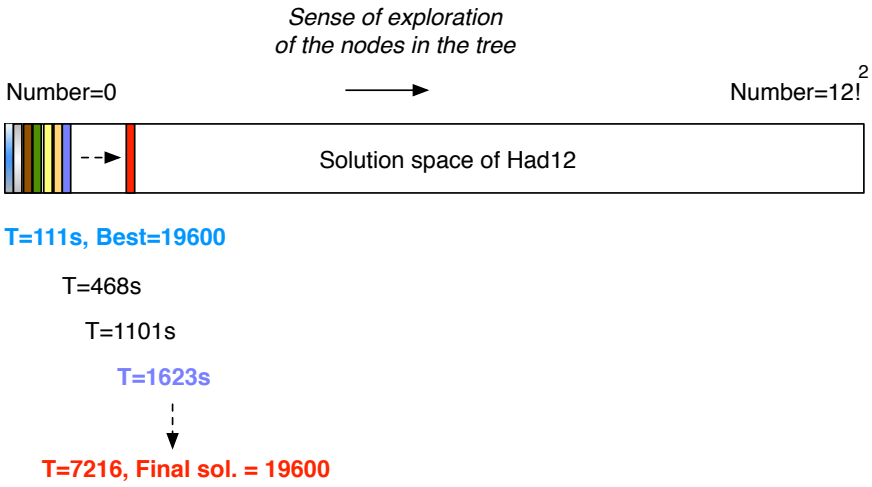


Figure 5.14: Search space of the Q3AP benchmark Had12, explored by the B&B algorithm when used alone during 2 hours.

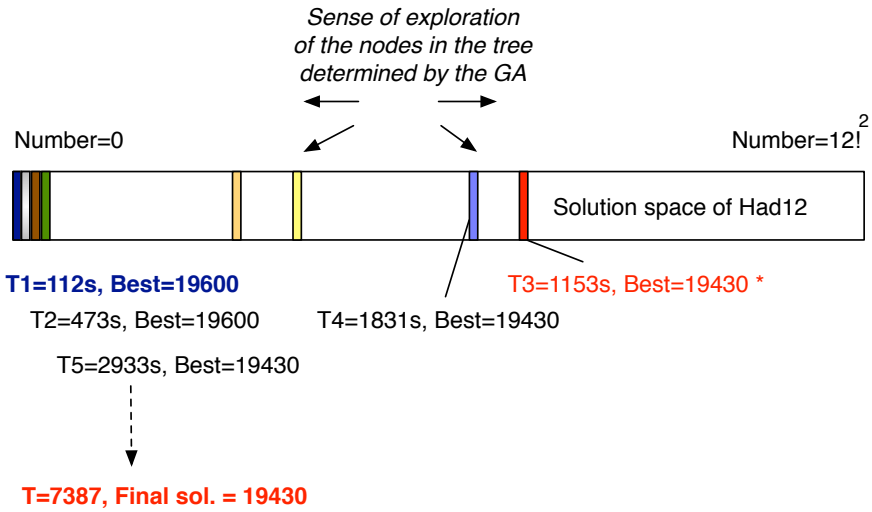


Figure 5.15: Search space of the Q3AP benchmark Had12, explored by COBBIGA after 2 hours run.

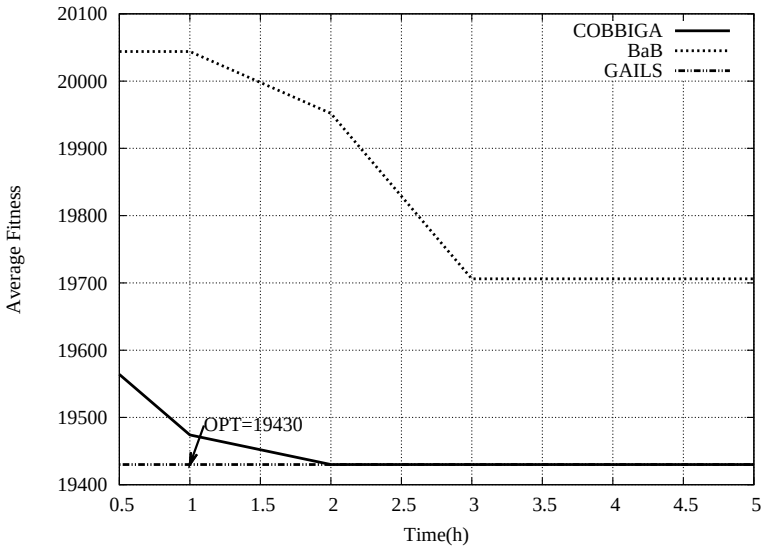


Figure 5.16: Average fitness values obtained by the algorithms COBBIGA, B&B and GAILS for the benchmark Had12.

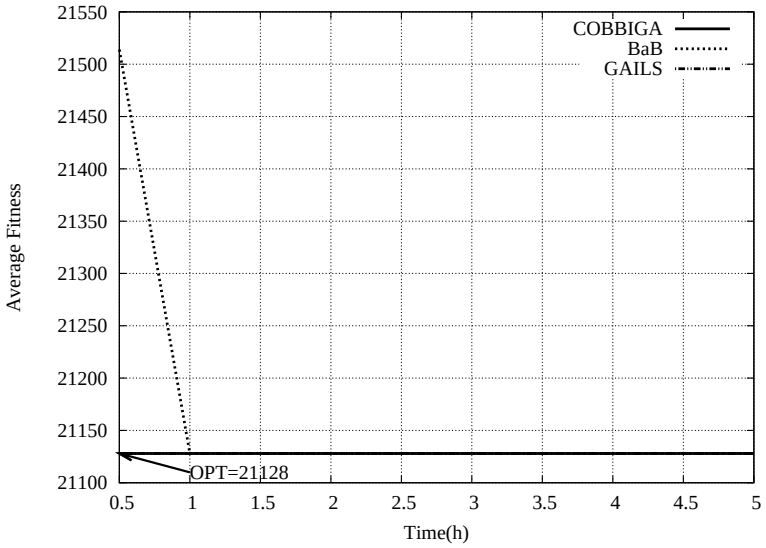


Figure 5.17: Average fitness values obtained by the algorithms COBBIGA, B&B and GAILS for the benchmark Had12b.

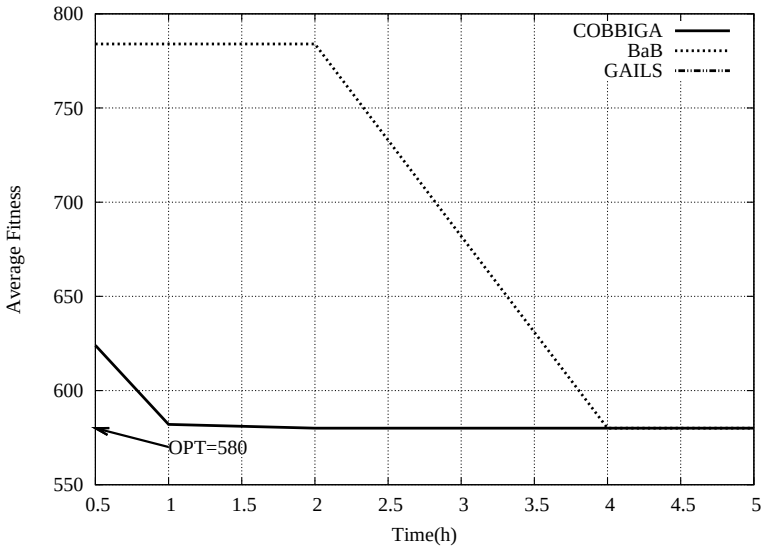


Figure 5.18: Average fitness values obtained by the algorithms COBBIGA, B&B and GAILS for the benchmark Nug12.

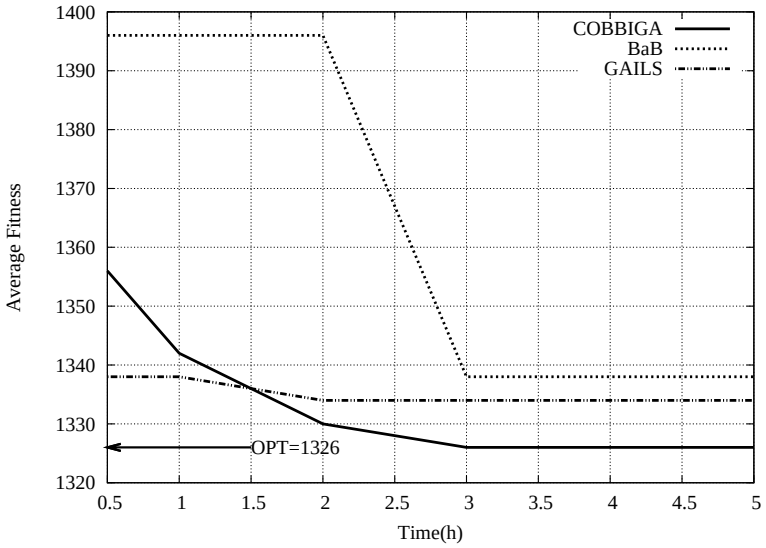


Figure 5.19: Average fitness values obtained by the algorithms COBBIGA, B&B and GAILS for the benchmark Nug12c.

Chapter 6

Parallel hybrid optimization methods for permutation-based problems

Contents

6.1 Introduction

It is acknowledged that hybridization schemes combining a variety of search algorithms are more powerful than standard optimization techniques when used separately. However, the required computational effort is also increasing. Thus, parallelization is a way to achieve better processing time and also to get better solution quality in some parallel models (see Section 2.5). In this chapter, we start by introducing the target parallel architectures, the computational grids. Then, we discuss the advantages of special parallelization strategies used both in metaheuristics and exact search algorithms: space decomposition techniques

in which the search space is decomposed on several sub-spaces and shared among parallel cooperating search agents. In this thesis, this parallelization strategy is applied to the hybrid schemes COBBGIA and HGABB/HAGABB presented in the previous chapter. The obtained parallel algorithms are experimented on large benchmarks of Q3AP.

6.2 Computational grids

Since the 90^{th}, distributed systems have gained a great development both in terms of software and hardware. Indeed, the development of high-speed networks and the generalization of the Internet lead to the emergence of a new generation of distributed systems and a new interpretation of the concept of resource sharing used in traditional computer networks. After the emergence of the concept of *grapes* or *clusters*, which refer to network of workstations, the concept of *grid* appeared in the mid-90^{th}. It designates a meta-computing system which includes a large number of computing resources from different administration domains and geographically distributed, which are shared and virtually accessible for end-users to run CPU intensive computations.

More generally, the concept of grid can also include all the software services offered by a grid in order to facilitate its use by end-users: data security, fault tolerance, easy access to the resources with abstraction to the multi-domain of administration, etc. Many middlewares and management systems for grids including such services have been developed in the last two decades. For example, we can site Globus [32] and Condor-G [35]. The Globus project aims to provide a low-level toolkit including basic mechanisms such as secure inter-domain communications, high-speed remote data access, user identification interface, etc. The Globus low-level toolkit is used to implement high-level services for the grid. Condor-G is an extension for grids of Condor [78]. It includes both low-level mechanisms from Globus and high-level services from Condor. After the identification step, the user submits a job request to the system, the requested number of resources, if available, is allocated to the user. Then, the user's application is downloaded and Condor-G is responsible of distributing the tasks of the application among the allocated computing resources. For more details about computational grids and related technologies, please refer to [33], [62] and [57].

To conclude, in [8] a computational grid is defined according to four main characteristics:

- Multi-domain of administration: that is the resources are distributed among a set of administration domains which are managed using different administration policies. For example, some resources may be protected by firewalls.

- Heterogeneity of the resources: the resources in the grid may be heterogeneous both in terms of hardware and software. In terms of hardware, the grid may include different machine architectures with different specifications (memory size, processor frequency, etc). From the software point of view, the available operating systems and software libraries may be different.

- Volatility of the resources: this means the availability of the resources is variable

because some resources may be down or simply requested by their owners, if we suppose that the resources in the grid may not be dedicated. An example for such grids is SETI@home, the resources are personal computers connected through the INTERNET.

– Scalability: the grid is large scale and easy to expend and grow. The scale of the grid is large in terms of network communications and in terms of number of processing cores.

Those characteristics of a computational grid lead to a set of challenges for parallel optimization algorithms. The volatility of resources may lead to the loss of the optimal solution if no *fault tolerance* mechanism is used, in the case of exact algorithms. For metaheuristics, this is not a real challenge as we are not looking for optimal solutions. The multi-domain administration requires the use of adequate *deployment strategies* that allow to access all the resources. Finally, the heterogeneity of resources requires the use of standard tools and frameworks especially for the communication issue. For example, one of the standard communication schemes is MPI (message passing interface) which is a library specification for message-passing communication protocols.

On the other hand, the hardware heterogeneity includes different computing capabilities (memory and processor frequency). Thus, for a better use of the resources, a *load balancing* mechanism should be used in order to balance the computation load among the available resources. Finally, the large scale of the grid leads to higher network latency and requires mechanisms for communication optimization in the system. On the other hand, the use of a large number of computing cores may lead to a scalability issue in the parallel application. For example, if a client/server implementation is used, the number of simultaneous clients is limited in order to avoid to overload the server. Pair-to-pair and hierarchical implementations can also be used to gain more scalability.

6.3 Parallelization of COBBIGA and HGABB/HAGABB

In the flat portion of the taxonomy of hybrid metaheuristics proposed by Talbi in [102], it is distinguished between global and partial methods. In global methods, the entire solution space is searched simultaneously by the cooperating algorithms. Alternatively, in partial hybrid methods, each algorithm is responsible of optimizing a given sub-space of the solution space and the solutions of the different partial searches are combined to form a solution to the global problem.

The class of partial hybrid methods is also referred to in the taxonomy of cooperative hybrid methods proposed by El-Abd and Kamel in [29], as space decomposition techniques. This class of cooperative hybrid methods, as their name indicates it, uses space decomposition techniques to reduce the size of the search space to be explored by each cooperating agent. The global problem to solve, of size n, is decomposed into sub-problems of size $n - p$ in which p decision variables are already fixed. Because any cooperative hybrid method is also considered as a parallel method, this kind of techniques are very used in parallel metaheuristics as well and are considered in the classification of parallel metaheuristics proposed by Crainic and Toulouse in [38].

The objectives of such techniques can be summarized in two points:

– Similar to the space decomposition in tree-based exact algorithms such as B&B, the space decomposition in parallel hybrid metaheuristics may help to simultaneously optimize disjoint sub-spaces of the solution space. That is the diversification of the search to different sub-spaces.

– Avoid the optimization of the same decision variables by more than one search agent. Thus, the underlying parallel architecture is well exploited as no redundant search is carried out by the different cooperating agents, especially when the number of agents is significant.

In [24], Debels *et al.* proposed a metaheuristic for resource-constrained project scheduling with space restriction techniques. A decomposition approach that splits the problem instances into smaller sub-problems, solved by either an exact search or a heuristic, is experimented. The principle of this method is as follows. The algorithm starts by optimizing the problem of size n until a stopping condition is reached. Then, iteratively a sub-problem of size i ($i = 1, \ldots, n$) is created by fixing the positions from 1 to i in the solution vector. Then, the i^{th} sub-problem is optimized until a stopping condition is reached. Afterwards, it is placed in the original problem to likely achieve a better solution.

Van den Bergh *et al.* also reported a cooperative approach to particle swarm optimization in [105]. Multiple swarms are used in their algorithm which simultaneously optimize different components of the solution vector. Indeed, if the solution vector is n-dimensional, each swarm optimizes a 1-D vector and there are exactly n cooperating swarms. However, the evaluation function requires the n-dimensional vector. Thus, a context vector is carried out in which the best element for each component is stored. This vector is used locally to evaluate the component optimized by the swarm. This improved PSO, called CPSO, is experimented on mathematical functions and outperforms the standard PSO algorithm.

In this thesis, we also use space decomposition techniques to parallelize the hybrid GA-B&B schemes proposed in Chapter 5. However, for more scalability, our approach is based on parallelization techniques used for tree-based exact search such as B&B. As already mentioned in Section 2.5.1, one of the most successful parallelization strategies for B&B algorithms consists in distributing sub-trees to multiple threads executing the same B&B algorithm on different regions of the search space/tree. Compared to the space decomposition strategies used in metaheuristics, this is more adequate for enumeration-based exact algorithms. Indeed, exact algorithms require the use of a larger number of parallel search processes. Thus, the distribution of sub-trees is a more scalable approach. In this thesis, as we are interested in hybrid schemes between B&B algorithms and metaheuristics, it is interesting to reuse the parallelization strategies used for B&B in the context of parallel hybrid metaheuristics. Moreover, in the hybrid schemes COBBIGA and HGABB, a unified representation of the search space, a B&B tree and the concept of interval, is used both in the B&B algorithm and in the GA. Thus, the parallelization of the two schemes is simplified. In the following , we

present the parallelization strategies used for the two hybrid methods.

6.3.1 Parallelization of HGABB/HAGABB

The parallelization strategy used for HGABB/HAGABB is described in this section. Two parallel models are discussed. The first model is based on a traditional hierarchical master/slave model and the second one uses explicit space decomposition using the B&B tree and the concept of interval.

6.3.1.1 Hierarchical master/slave island model

In HGABB/HAGABB the generic GA is not altered by the hybridization with the B&B algorithm. Thus, standard parallel models for EC (Evolutionary Computing) could be applied to HGABB/HAGABB. As the underlying physical support of the experiments is a computational grid, a coarse grain parallel model (island model) is more convenient. On the other hand, the most expensive task in the hybrid algorithm is the transformation step because of the B&B solver embedded in the mutation operator. Thus, a master/slave model implemented on the top of each island will help to speed up the overall execution time of the algorithm. This model is illustrated in Figure 6.1.

This is a typically hybrid parallel GA in which a set of islands evolve multiple subpopulations at the highest level and each island uses slaves to parallelize the CPU intensive tasks of transformation (crossover+mutation) and evaluation. In addition, each slave uses a memory module to store optimally solved sub-spaces by the B&B solver, represented as intervals. This list plays the role of a tabu list: each time a new node to solve by the B&B is formed, it is checked if its range is not included in the tabu list in order to avoid very expensive and useless work redundancy. The communications between the slaves and the master and between the islands are asynchronous.

6.3.1.2 Space decomposition parallelization

By analogy to the parallelization strategies used for B&B, in this thesis we introduce some techniques used in parallel B&B algorithms to hybrid metaheuristics. In [76], the concept of interval is used to represent a contiguous set of nodes in the B&B tree. The global interval representing the whole search space is shared between the available processors. This space decomposition technique could be reused in the context of GAs. Moreover, In HGABB/HAGABB, as the B&B solver incorporated to the mutation operator works on sub-trees and intervals, it is more convenient to use space decomposition in the GA islands as well in order to diversify the search of the B&B to different and disjoint regions in the search tree. The decomposition by intervals has the advantage of being scalable unlike the decompositions used for metaheuristics in the literature, in which the number of concurrent searches is limited to the size of the decision vector.

A parallelization strategy for HGABB/HAGABB based on space decomposition is depicted in Figure 6.2. Unlike the traditional hierarchical model, in this model each island

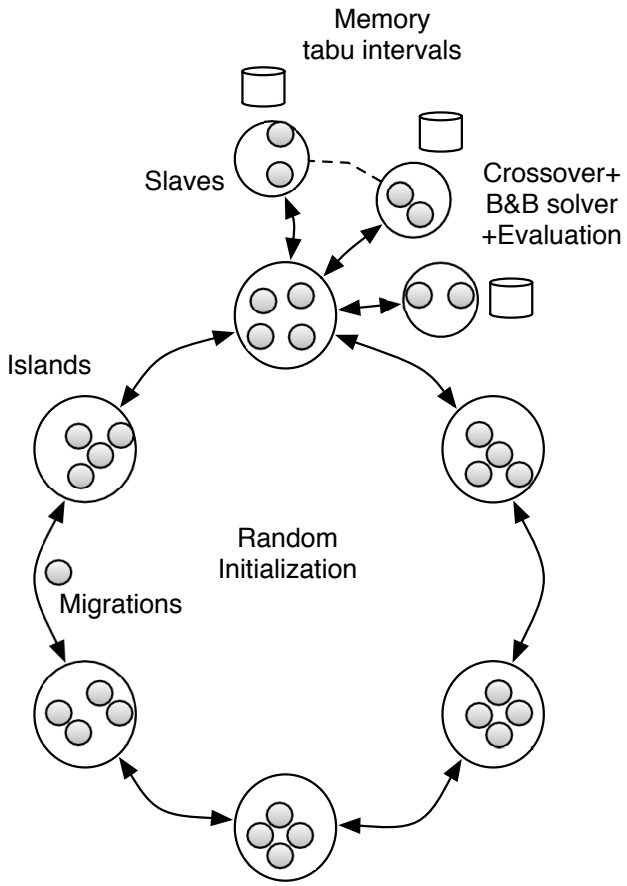


Figure 6.1: Hierarchical parallel island model for HGABB/HAGABB.

evolves a sub-population initialized from a unique disjoint sub-interval. The interval initialization is used to diversify the search to different subtrees in order to provide the B&B solver (embedded in the mutation step) with tree nodes uniformly distributed in the tree. The islands are disconnected from each other and each island is limited to its allocated sub-space until the stopping criterion of the application is reached. In order to guarantee that each island will be limited to the allocated interval, I-GA (interval-based GA) instead of standard GA is run in each island. Each island uses additional resources as slaves in order to efficiently run the transformation and evaluation steps. As in the previous model, each slave instantiates a hybrid mutation operator only once and uses a memory module in order to store the sub-intervals solved to optimality by the B&B solver. In the following, the initialization technique and the cooperation between islands are detailed.

Initialization of the islands

In this model, the GA islands are initialized using the interval-based initialization method introduced in Section 4.5. As already mentioned, since each island is using a B&B

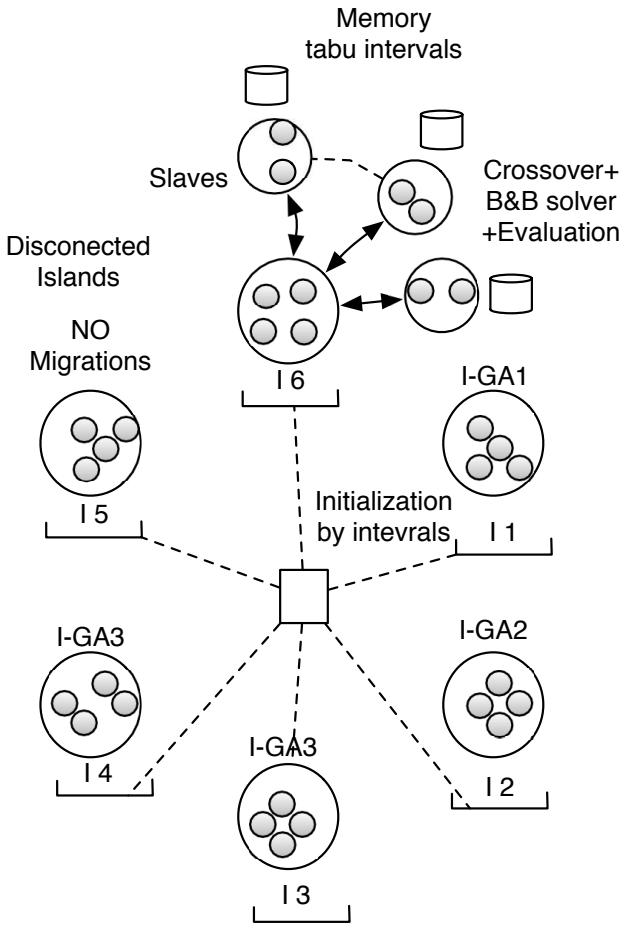


Figure 6.2: Hierarchical parallel island model for HGABB/HAGABB with interval-based space decomposition among islands.

solver incorporated to the mutation operator, one may take advantage if these populations are not interlaced in order to avoid redundant search in the B&B. This makes each island searching in a different region of the search space. An illustration of this island initialization procedure is shown in Figure 6.3. This is an example for a problem using one permutation of size $n = 4$. The idea is very simple. First, we divide the global interval into d sub-intervals, d is the number of islands (three islands in our example). Then, each island will use the interval-based initialization method to a restricted sub-interval. Notice that if we use a ring topology, adjacent islands will have adjacent initial populations regarding the tree representation.

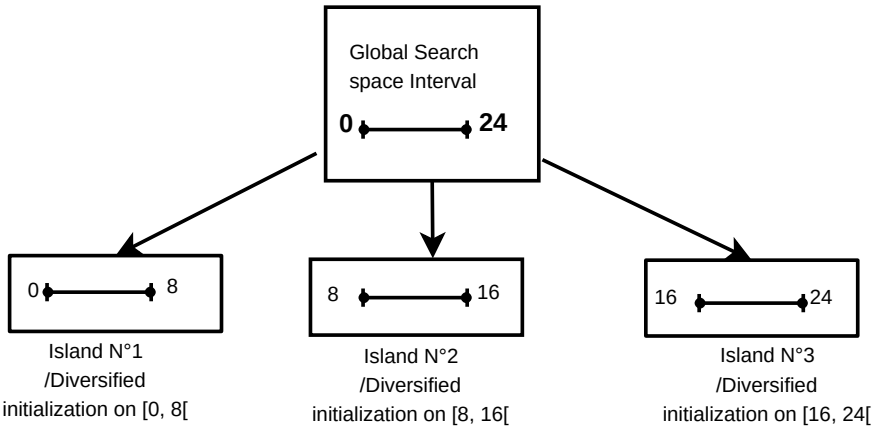


Figure 6.3: Interval-based island initialization.

Clusters of cooperative islands

After the initialization step, the islands can still continue working as in the standard island model with standard transformation operators and exchange migrants in order to diversify the populations in each island. Or, as depicted in Figure 6.2, each island uses interval-based transformation operators in order to limit its search space to the given interval. That is, a I-GA is run instead of a standard GA in each island and there are no migrations between islands working on different intervals.

Nevertheless, it has been pointed out in many works that GAs with migrations achieve better results in comparison to the sequential algorithm. Indeed, the synergy between the cooperating islands makes the global search more efficient. Thus, even if space decomposition techniques are used among islands, clusters of standard island model can be implemented at the upper level of the model in order to explore each sub-space (sub-interval) efficiently. The global solution of the problem is determined at the end of the computation, the local best solutions of all the clusters are merged and the best solution among all is chosen as a global optimum for the problem.

An illustration of this model is depicted in Figure 6.4. In each cluster, a cooperative island model is run on a restricted sub-space delimited by the allocated sub-interval. Each island of a given cluster runs a I-GA on the sub-interval allocated to the cluster. Islands of the same cluster cooperate to efficiently solve the sub-interval.

6.3.2 Parallelization of COBBIGA

The parallelization of COBBIGA is also based on hierarchical master/slave island model with interval-based space decomposition. Indeed, the main task in the GA islands is the transformation step as it incorporates a local search algorithm. Alternatively, in the B&B islands, the main task is the B&B solver. Thus, a hierarchical parallelism with two levels helps to speed up the execution time of both B&B solvers and GAs. Moreover, in COBBIGA, the global search is guided by the B&B tree. Thus, the interval-based explicit space

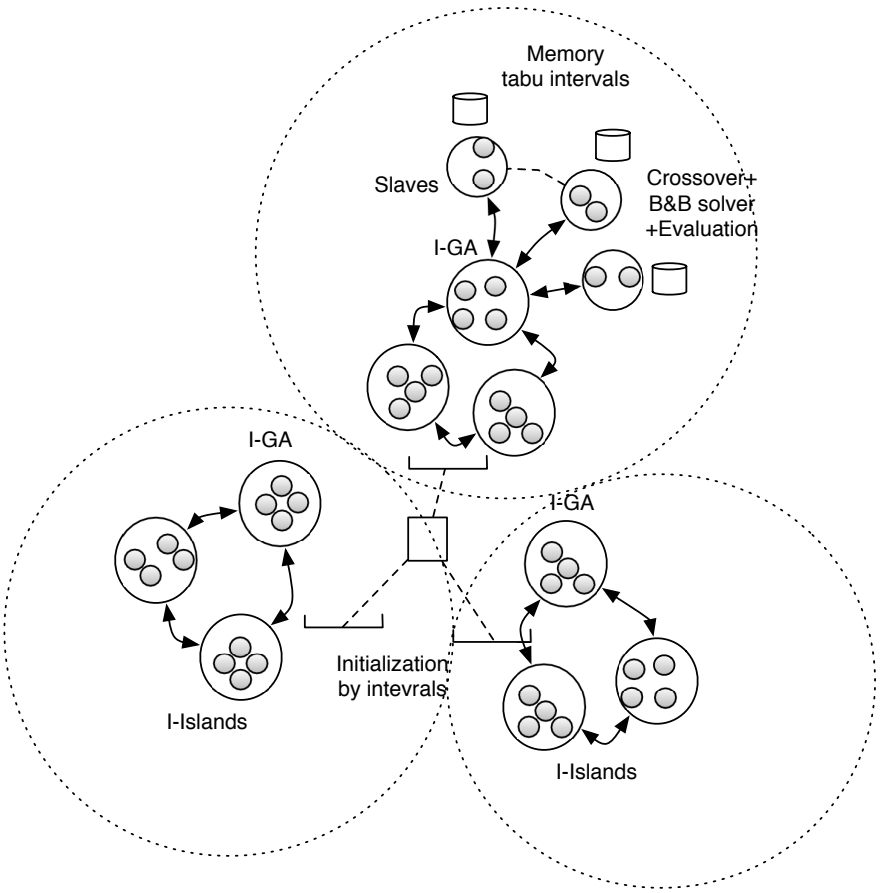


Figure 6.4: Clusters of hierarchical parallel I-GAs used to parallelize the hybrid algorithm HGABB/HAGABB.

decomposition is very recommended for COBBIGA in order to avoid the redundancy in the global search and to explicitly diversify the global search. The details are presented in the following.

6.3.2.1 Two levels of parallelism

The first level consists in using the master/slave model in each island (B&B and GA). Each GA island uses additional resources (slaves) in order to parallelize the evaluation and the transformation tasks, while the B&B island uses additional resources in order to solve a group of nodes simultaneously. Each B&B slave includes a B&B solver that simply receives one node to be solved and returns back the result to the B&B island. This level of the parallelism is illustrated in Figure 6.5.

The second level of parallelism is illustrated in Figure 6.6 and consists in dividing the global search space, represented by the global interval I, into k equal sub-intervals

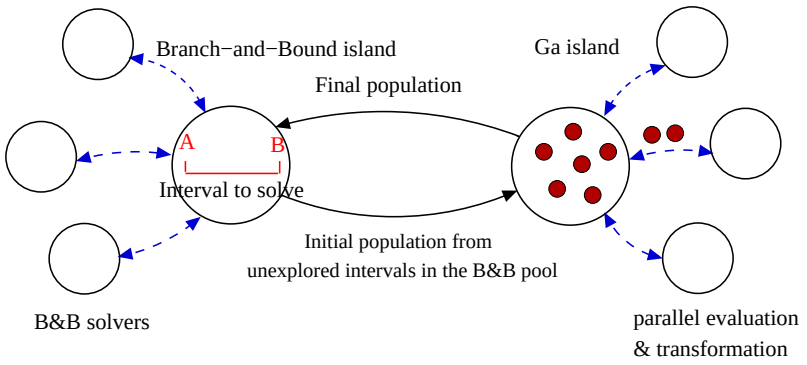


Figure 6.5: The first level of parallelism in COBBIGA: a master-slave model on the top of each island.

$\{I_1, I_2, ..., I_k\}$. These sub-intervals will be distributed among k couples of islands B&B-GA. This creates a kind of hierarchical master/slave model. The B&B islands in all the couples B&B-GA are connected to each other via a complete topology in order to exchange their best found solutions and balance their loads in case one of the B&B islands has no more work to do. On the other hand, the GA islands do not exchange data because each GA is working in a different sub-space.

This second level of parallelism is added in order not to overload the islands in case of large scale experiments (large number of slaves communicating with a single master). The other reason is to take advantage from the parallelization power inherited from the tree-based representation and the interval coding proposed in [76]. Indeed, while sharing the global interval among the k couples B&B-GA, the couples will be given the possibility to search simultaneously in different regions of the huge search space.

6.3.2.2 Tuning of the number of couples B&B-GA

The number of couples B&B-GA in COBBIGA should not be too large. Indeed, the B&B islands are connected through a complete topology. Each time a B&B island finds a new best solution, this solution is broad-casted to all the other islands. Thus, a big number of B&B islands will generate too much communications in the network.

Alternatively, in order to have sub-intervals which contain only nodes of the same depth d, the chosen number of B&B-GA couples should be a divisor of the number of nodes in the depth d of the tree (see Equation (6.1)). In Equation (6.1), n is the size of the problem and m is the number of permutations.

$$TOT\_ISLANDS = \{x/x \ \ is \ \ a \ \ divisor \ \ of \ \ [n \cdot (n-1) \cdot (n-2) \ldots (n-d)]^m\} \qquad (6.1)$$

For example, if we consider a 2-permutation problem of size $n = 14$, the number of nodes in the first level of the tree is 14^2. The number of islands (and sub-intervals) should be a divisor of 14^2 i.e. 2, 7, 14, 28, 196 in order to have an equal number of nodes with depth 1

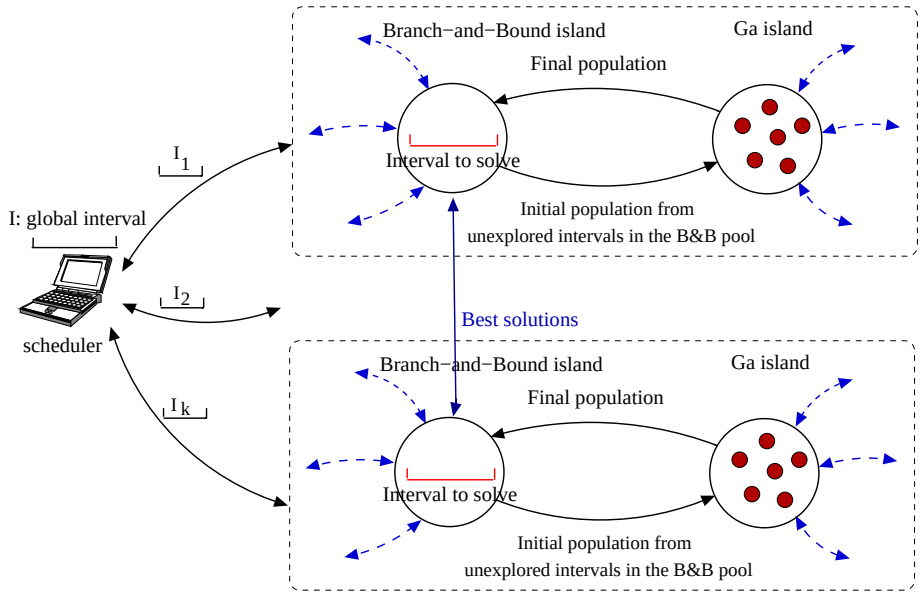


Figure 6.6: Topology of the parallel COBBIGA: each GA island is connected to one B&B island. The B&B islands are completely interconnected and the GAs islands are independent.

in each island. The motivation for this kind of static decomposition of the search space is that if the interval is shared regardless of the structure of the tree, when the sub-intervals are transformed into tree nodes, one may obtain a large number of nodes which are close to the leaves (nodes with a large depth). Those nodes will all be distributed among the slaves to be solved by the B&B solvers. Nevertheless, some of such nodes might be located in a larger sub-tree which could be eliminated at a higher level in the tree.

6.3.2.3 Load balancing between the B&B islands

The load balancing is partially performed during the initialization step because the islands get equal sub-intervals. At the end of the search, if one of the B&B islands have no more nodes to solve, a job request is gradually sent to the neighbors to get half of the remaining intervals of one of them. If no answer is received after r requests (r predefined by the user), the B&B island terminates its process. The search is stopped when all the islands have no more nodes or when the time limit is reached. The stopping operation is done by a special process named the *scheduler*. The role of the scheduler is the allocation of the machines to the different types of processes. Once the global interval is partitioned among islands, the new unfold operator presented in Section 5.3.4 is used in order to transform each interval into a pool of nodes in each island.

6.3.3 Limits of the hierarchical island model when applied to exact optimization

If the objective of the search is to prove the optimality of the best found solution, the hierarchical parallel model based on the island model is not suitable, especially for large scale

experiments. Indeed, in the island model, the number of islands is determined statically as well as the number of individuals in the population. While in parallel B&B, the number of B&B processes should be determined dynamically and adapted to the number of available resources especially when it is executed in volatile environments such as computational grids. Nevertheless, if the parallel COBBIGA is used as a metaheuristic, the hierarchical parallel model based on the island model is well suited. Indeed, it facilitates the communication with GAs and reduces the effort of implementation using existing softwares for parallel GAs.

6.4 Framework coupling-based implementation

The different frameworks used to implement the parallel hybrid algorithms, the underlaying physical support of the experiments and the results obtained on standard Q3AP benchmarks, are detailed in the sequel.

6.4.1 Q3AP benchmarks and related works

Owing to the similarities between Q3AP and QAP, many successful metaheuristics and exact search techniques applied to QAP have been generalized and adapted for Q3AP.

The only exact method used for this problem in the literature is the B&B algorithm proposed by Hahn and Kim [58] and a parallel version for shared memory architectures proposed by Galea and Le Cun [40]. The lower-bound technique proposed for Q3AP in [58], is based on Lagrangian relaxation of the level 1 RLT (reformulation linearization technique) of the problem. This lower-bound is derived from the lower bound proposed by Hahn for QAP. An interesting recent review of exact techniques for assignment problems (QAP, Q3AP, GQAP, etc) is proposed by Hahn *et al.* in [48]. In this review, emerging quadratic assignment problems are studied. The most recent and successful exact techniques from the literature used for each problem are reported as well as the largest benchmark solved to optimality. For Q3AP, the largest benchmark solved to optimality reported in [48] is of size 14. In [47], optimal solutions for the Q3AP benchmarks Nug12 and Nug13 (derived from QAP benchmarks of the same name) are found using a sequential B&B algorithm based on the level 1 RLT lower-bound. Recently, Galea and Le Cun in [39], implemented a parallelization of Hahn's sequential algorithm for shared memory machines, coupled to the parallelization of the lower-bound itself. This parallelization leads to significantly improved times for the benchmarks Nug12 and Nug13, and optimal solutions are found for the benchmarks Nug14, Nug15 and Nug16a pushing the limit of optimally solved benchmarks for this problem to 16.

The first metaheuristics used to solve Q3AP are also reported by *Hahn et al.* in [47]. These algorithms are derived from the most successful metaheuristics used to solve QAP (simulated annealing algorithm (SA), fast ant colony algorithm (FANT), robust tabu search (RTS) and iterated local search (ILS)). The four algorithms were experimented using six Q3AP instances including two real instance, a size 8 and size 16 symbol-mapping problems. The remaining benchmarks are derived from QAP instances from the QAPLIB [14]: Nug12, Nug13, Nug15 and Thai9. The experimental results reported in [47] show that the Q3AP

instances of size 8 and 9 are solved to optimality in few seconds by the four algorithms. The best high quality solutions are obtained by ILS after one hour run for the benchmarks Nug12, Nug15 and for the real instance of size 16, and by TS for the benchmark Nug13. After 10 hours run, optimal solutions were found by both ILS and TS for Nug12 in the 8 considered runs, and by three algorithms: ILS, TS, SA for the benchmark Nug13. While the best known solution for Nug15 and for the real instance of size 16, were exclusively found by ILS.

Parallel hybrid genetic algorithms with local search mutations have also been used in some works to solve larger benchmarks. In [71], a hybrid GA combining a GA and a simulated annealing (SA) algorithm is proposed. Q3AP benchmarks of size 18 and 22 (Nug18 and Nug22) were addressed for the first time. In order to speed up the execution time of the metaheuristics applied to Q3AP, some authors used graphic processor implementations such as in [104] where a GPU-based Iterated Tabu Search using an extended 2-exchange neighborhood is proposed. Also in [69], a hybrid evolutionary algorithm for Q3AP is implemented for Many-Core Graphics Processors.

6.4.2 Implementation issues

The different algorithms proposed in this thesis were implemented using two different frameworks. The metaheuristic algorithms (ILS, HC, GA) are implemented using the ParadisEO framework [17]. While the B&B algorithm used for Q3AP is kindly provided by Galea and Le Cun from Versailles university (France) [40]. This algorithm is developed using the BOB++ framework [27]. The hybrid methods COBBIGA and HGABB/HAGABB are implemented using a coupling of the two frameworks. Indeed, a hybridization framework integrated to ParadisEO is developed in order to facilitate further implementations of hybrids combining exact methods developed using other C++ frameworks with metaheuristics developed using ParadisEO. The two basic frameworks BOB++ and ParadisEO, and the new hybridization framework are presented in the following.

6.4.2.1 The ParadisEO framework

As already mentioned, all the metaheuristics proposed in this thesis were implemented using ParadisEO (PARAllel and DIStributed Evolving Objects) [17] which is a C++ open source framework dedicated to the reusable design and implementation of parallel hybrid metaheuristics. ParadisEO provides a broad range of features for solving optimization problems, such as evolutionary algorithms, local search methods, different hybridization mechanisms for metaheuristics, etc. It also includes the most common parallel and distributed models, portable on distributed-memory machines and shared-memory multiprocessors, as they are implemented using standard libraries such as MPI (MPICH-II) and PThreads.

Figure 6.7 is an illustration of the different layers which compose ParadisEO. There are four main layers: EO, MO, MOEO and PEO. EO (Evolving Objects) includes evolutionary algorithms and related techniques. MO (Moving objects) contains local search techniques and single-solution metaheuristics (simulated annealing, tabu search, iterated local search, etc.).

MOEO (Multi Objective Evolving Objects) contains different metaheuristics and techniques for multi-objective optimization problems. Finally, PEO (Parallel Evolving Objects) offers basic tools for the easy implementation of parallel hybrid metaheuristics for both shared and distributed memory systems.

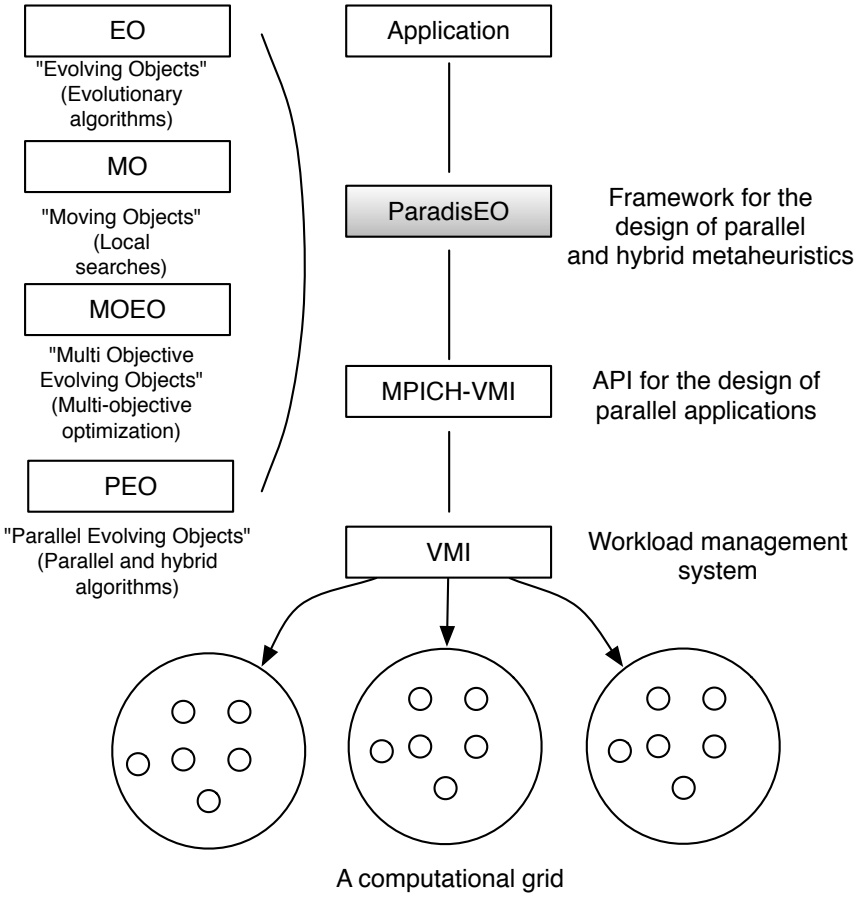


Figure 6.7: Layers of the framework ParadisEO.

6.4.2.2 The BOB++ framework

The B&B algorithm used in our experiments for Q3AP is developed using the BOB++ framework, this algorithm could be found in [40]. BOB++ [27] is an open source C++ framework dedicated to the design of exact methods for solving combinatorial optimization problems on parallel architectures. The parallelization is done using POSIX threads. However, the framework is easily expendable and other parallelization models are under development. Moreover, BOB++ provides many search algorithms (such as B&B, divide and conquer,...) applied to many problems (e.g. TSP, QAP, and Q3AP). Figure 6.8 is an illustration of the different layers of BOB++. To use this framework, the user must define the following classes which are problem specific:

– The Node class: represents a solution or a subproblem of the problem and contains the bounding operator.
– The Instance class: contains all the required input data for solving the problem.
– The GenChild class: contains the branching operator.

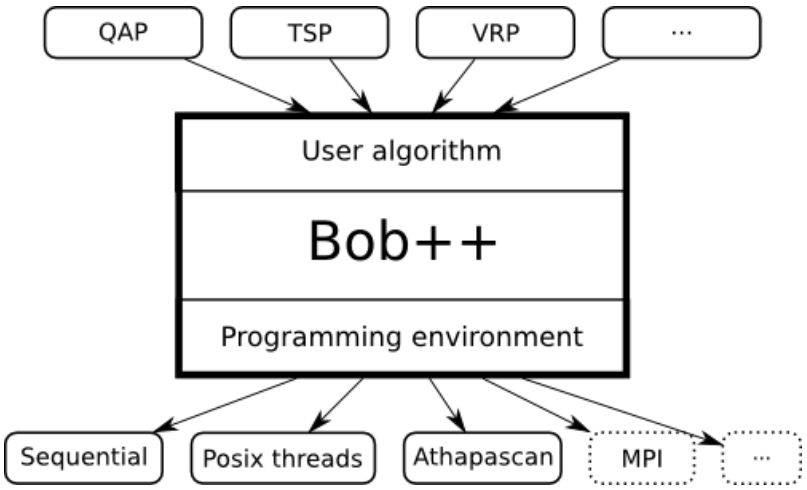


Figure 6.8: Layers of the framework BOB++.

6.4.2.3 ParadisEO- HEM: a hybridization framework for the coupling of exact methods and metaheuristics

In order to facilitate the implementation of any kind of hybridization schemes combining optimization methods developed in two different frameworks (here BOB++ and ParadisEO), we developed a hybridization framework integrated to ParadisEO, called Paradiseo-HEM for Hybrid Exact Metaheuristics. This framework contains template classes that can encapsulate external tree-based exact algorithms in a paradisEO-like algorithm.

The class diagram of this framework is shown in Figure 6.9. The light pink box classes are super classes used in ParadisEO to implement basic components common to all metaheuristics: *Service* is used by the communication interface, *moAlgo* is the super class for all local search algorithms and mutation operators, and the template parameter *EOT* is the basic class for a problem's representation. The *moAbstractNodeSolver* class is the basic class for exact solvers. Thus, all exact methods integrated to ParadiseEO must be sub-classes of the *moAbstractNodeSolver* class. For example, the class *Q3APBBNodeSolver* encapsulates the B&B solver for Q3AP developed using BOB++. The *peoPopSolver* class is used to solve a group of nodes by the B&B solver, either in a sequential (*peoSeqPopSolver*), or a parallel way (*peoParaPopSolver*). The *peoParaPopSolver* component is used to implement a hierarchical parallelism on the top of each B&B island according to the master-slave model. Finally, the *peoLoadBalance* class is used to implement a load balancing strategy which consists in distributing the remaining search space between neighbors.

Different hybridization schemes between tree-based exact methods and metaheuristics could be easily implemented using this framework for any other class of problems. For example, a cooperative hybrid algorithm between a B&B algorithm and a GA can be seen as an island model with two islands exchanging information synchronously or asynchronously. As the implementation of the island model is already provided by ParadisEO, the encapsulation of the B&B solver in an *moAbstractNodeSolver* object (which is a sub-class of *eoAlgo*, the same super class for a GA), is sufficient to build a cooperative hybrid between the two algorithms B&B and GA. Another possibility of hybridization consists in using the *moAbstractNodeSolver* to replace the mutation operator in a GA by a B&B solver (low-level hybridization). Indeed, the *moAbstractNodeSolver* class is a sub-class of the class *moMonOp* which is the same class used in ParadisEO to implement mutation operators. Finally, the language used to implement the B&B algorithm (C++) being the same than the one used in ParadisEO, this also helps to integrate the two frameworks more easily.

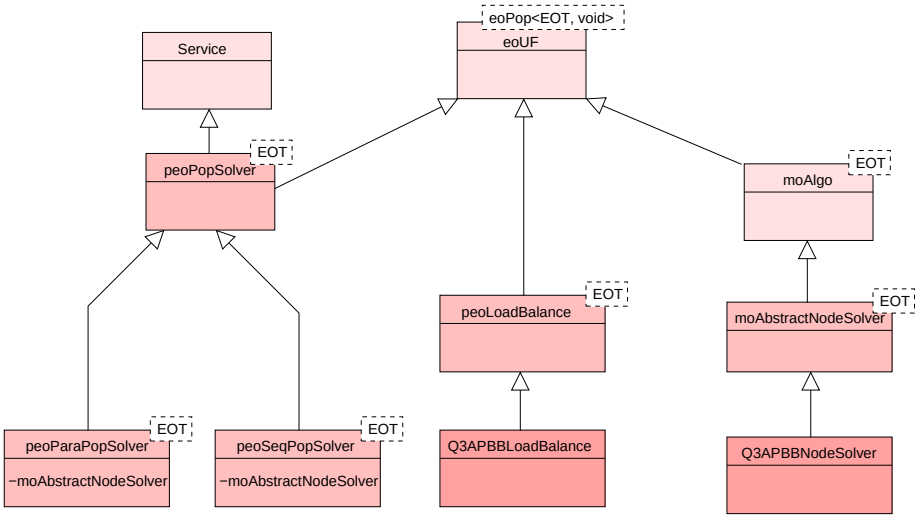


Figure 6.9: The UML class diagram of ParadisEO-HEM, the framework that enables users to couple external exact methods with metaheuristics provided by ParadisEO to build efficient parallel hybrid methods.

6.4.2.4 Adaptation of the platform B&B@Grid to Q3AP

This work is done in collaboration with M. Mezmaz (university of Mons, Belgium). The framework B&B@Grid for parallel B&B on grid, developed during the P.hd thesis of Mezmaz [77], is adapted to the Q3AP.

B&B@Grid has been initially experimented on large instances of the permutation flow-shop problem. In this thesis, an adaptation is done for Q3AP in order to prove the optimality of the solutions found by the hybrid algorithms HGABBB/HAGABB and COBBIGA. Since the parallel COBBIGA algorithm is also based on the B&B tree, the search space explored by COBBIGA is recorded (as a list of intervals). In order to save time, this search space is ignored during the process of implicit enumeration done by a B&B algorithm alone. Mainly,

the B&B@Grid is initialized with the remaining intervals and the best found solution from the previous executions of COBBIGA. The components of the B&B algorithm for Q3AP used in B&B@Grid are detailed in the following.

Decomposition

The decomposition is done in such a way to produce a lexicographic permutation tree. A list of forbidden elements in the incomplete solution is maintained for each permutation (elements that appear in the fixed part of the solution). The next position to fill in the two permutations of the Q3AP solution are filled with the remaining elements. Mainly, the next position in the first sub-node is filled with the smallest element among all the remaining elements, the second sub-node is filled with the next element in a numerical order, and so on.

Bounding

The bounding function used in B&B@Grid for Q3AP is the lower-bound proposed by Hahn *et al.* [47] and the implementation is provided by Galea and Le Cun [40] (it is the same bound used in the BOB++ B&B solver [40]).

Selection

Because of the use of the intervals, the selection strategy is necessarily depth first selection. In [76], the B&B@Grid is applied to the flowshop permutation problem in which the bounding function is not CPU time intensive. Thus, every time a tree node is decomposed, its sub-nodes are inserted in a priority queue in which the nodes with the best lower-bound are solved first. Thus, the numbers in the tree are associated to the nodes according to their arrangement in the pool of nodes instead of the lexicographic order of permutations. Nevertheless, in the case of Q3AP, the bounding function is too expensive both in terms of CPU time and in terms of memory consumption. Thus, the nodes are solved in a default order and because of the cooperation with the GA, the numbers in the tree are affected to the nodes according to the lexicographic order of permutations. Indeed, the B&B@Grid is used to solve the remaining intervals from the execution of the hybrid algorithm COBBIGA and the nodes in COBBIGA are numbered according to the lexicographic order. Thus, the same order should be used in B&B@Grid.

Deployment paradigms on the grid

The deployment paradigms supported by B&B@Grid are both centralized farmer-worker and P2P paradigms. The two versions are based on the concept of interval. In the experiments we did for Q3AP, we used the centralized farmer-worker deployment approach. Two types of processes are used: farmer and worker. The farmer is the unique manager of the computation, it is hold by a dedicated machine. On the other hand, each worker holds a B&B algorithm which is used to explore a given region of the search space represented as an interval. A set of workers explore different regions of the search tree simultaneously. The farmer maintains a list of unsolved intervals, which initially contains a single interval (the global interval). The workers are hosted by additional resources which join the computation following the strategy

of idle CPU cycles. This means that the resources holding the workers are not necessarily reserved in advance. This type of strategies is well suited to computational grids in which the machines are not necessarily dedicated to the user. Check-pointing and load balancing mechanisms are provided by B&B@Grid. The scalability of this centralized approach has been experimented in [77] and the results of the simulations indicate that the farmer can support more than thousands of workers without being overloaded. In this thesis, B&B@Grid is used with the farmer-worker paradigm in order to prove the optimality of some solutions found by the hybrid algorithms. The parallel COBBIGA algorithm is used for a given period of time and the remaining intervals are solved by B&B@Grid.

6.4.3 The hardware support of the experiments

All the experiments presented in this thesis were conducted over Grid'5000 which is an international experimental grid that today inter-connects 10 sites in France via RENATER (the French academic [1] network), one site in Luxembourg via RESTENA (the luxembourgish academic network [2]) and one site in Porto Algere (Brazil) (see Figure 6.10).

Currently, the GRID is composed of more than 6000 cores with more than 100 Tb of non-volatile storage capacity. The inter-connections sustain communications of 10 Gbps. The objective of Grid'5000 is to provide an experimental large scale platform for researchers working in the field of parallel and distributed applications. Grid'5000 is built in such a way to facilitate the deployment of user applications, and the retrieval of the results in a transparent way for the end user. For this, many management systems, middlewares and services were developed by different collaborating laboratories. Some tools are still under development while others became part of the grid and are used during the different steps of the process of experimentation. The two principle Grid'5000 tools are OAR [3] and Kadeploy [4].

Kdeploy is a useful tool which enables the users to create and personalize their own experimental environments. It allows the deployment of a complete virtual user image on the reserved machines. A user image may include an operating system (exp. Linux), parameters for the user environment (variables), special libraries required by the user's application, etc. However, if the user's application do not require special softwares, the user may also use the default image available in the machines.

OAR is a batch system responsible of the management of the resources in the sites. The provided services include among others: submitting of job requests (reservation of a specified number of machines), access to different informations about the sites, the clusters and the nodes (exp. number of free/occupied/down nodes in each site), withdrawing a reservation, etc. In order to allow all the users to have moderate access to the grid, some scheduling roles are implemented in OAR, in addition to a user Charter. Indeed, there are different job submission modes. Among them we quote the INTERACTIVE, the BATCH and the BESTEFFORT modes. In the interactive mode, the machines are immediately reserved

1. http://www.renater.fr
2. http://www.restena.lu/
3. http://oar.imag.fr
4. http://kadeploy.imag.fr

and the user is logged to the first node in the list of allocated resources. Alternatively, the BATCH mode is used to specify a further time for the starting of the reservation. Finally, the besteffort mode uses a different queue for the job requests. The required number of machines is allocated to the user only when no other regular job uses or requests them. Multi-site experiments are also possible using a grid version of OAR, OARGRID. The latter enables the reservation of CPUs from more than one site. For a more detailed presentation of Grid'5000 please refer to [22].

The machines we used in the experiments presented in this chapter belong to 3 sites in Grid'5000: Nancy, Sophia-Antipolis and Grenoble. The used clusters are Griffon (64 bits, 4 Dual core CPUS, 2.5 GHZ.) and Grellon (64 bits, 2 Dual core CPUS, 2.0 GHZ.) in Nancy, Suno (64 bits 4 Dual core CPUS, 2.26 GHZ), Sol (64 bits 4 Dual core CPUS, 2.6 GHZ) in Sophia and the clusters Genepi and Edel (64 bits, 4 Dual Core CPUs, 2.5 GHZ) in Grenoble.

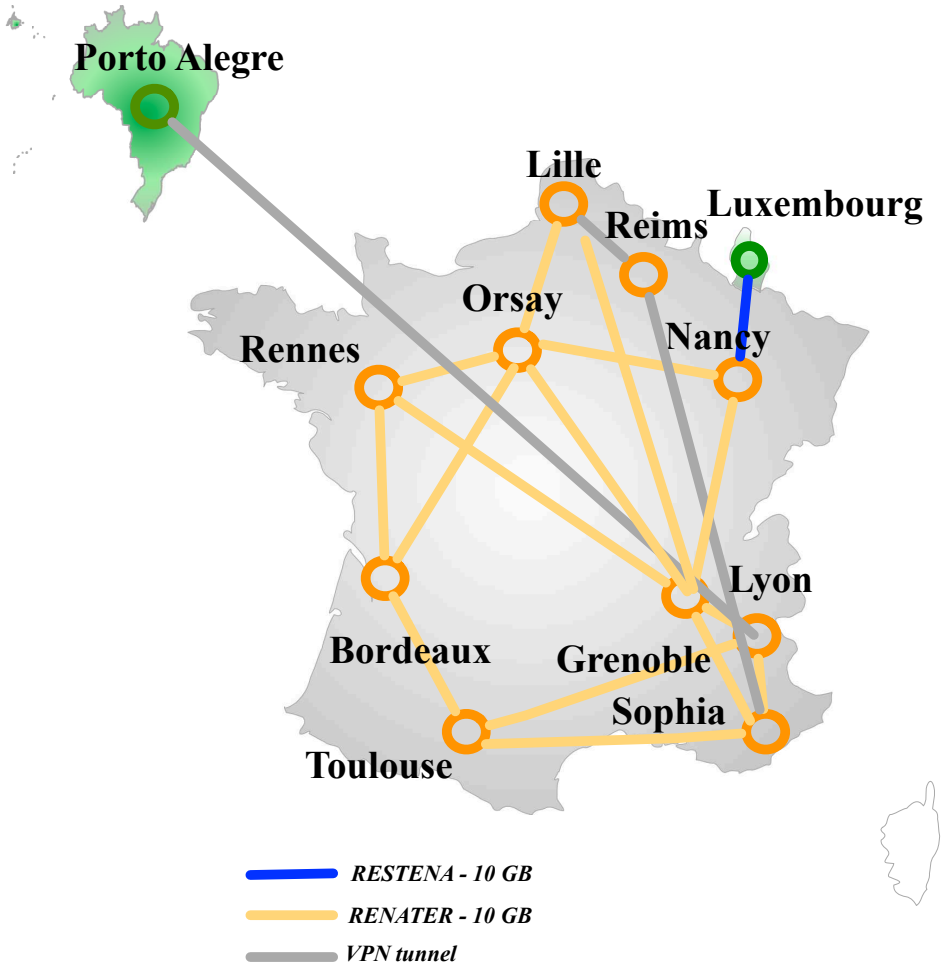


Figure 6.10: The experimental computational grid Grid'5000.

6.4.4 Experimental results

In this section, we will present the results of the experimentations conducted on standard Q3AP benchmarks. The focus is mainly set on the parallel COBBIGA algorithm when solving large Q3AP benchmarks. The experiments are divided into three groups according to three objectives:

- Evaluation of the impact of the hybridization with GA islands on the search of the B&B islands.
- Solving some benchmarks to optimality using the intermediate results obtained using COBBIGA.
- Evaluation of the performances of the parallel COBBIGA as a metaheuristic against a standard hybrid metaheuristic (a parallel hybrid GA-ILS algorithm).

The Q3AP benchmarks used to evaluate the performances of the parallel COBBIGA are: Nug14, Had14, Nug14b, Had15, Nug15 and Nug16a. All derived from standard QAP benchmarks which could be found in the QAPLIB [14]. Some benchmarks are generated from other instances of larger size in the QAPLIB: Had15 is generated from Had16 (one column and one row form each matrix of Had16 are withdrawn) and Nug14b is obtained by inversing the distance and flow matrices in the standard benchmark Nug14.

6.4.4.1 Evaluating the performances of the parallel COBBIGA against parallel B&B on a computational grid

In this section, we evaluate the impact of the hybridization with GA islands on the final solution found by the B&B islands in COBBIGA. For this, the parallel algorithms COBBIGA and B&B were applied to the benchmarks cited above. Both methods were given equal execution time periods (2 hours or 5 hours for large instances) to solve each benchmark using the same parallel hierarchical master-slave model on the same number of machines.

The parameters used in COBBIGA are listed in Table 6.1. The migration frequency (parameter MIG_FREQ) between each couple of islands (B&B and a GA) was fixed to 60 generations: every time the GA performs 60 generations from its initial population, it sends its final population to the B&B island and gets a new initial population.

The parameter $NODE\_SIZE$ which determines the size of the nodes to be solved by the B&B solver in each island, is fixed to 10 for all the benchmarks except for Nug18 and Nug20. Indeed, the optimal values for the parameters $NODE\_SIZE$ and MIG_FREQ have already been determined in Chapter 5. For the two largest benchmarks Nug18 and Nug20, the size of the nodes to be solved is fixed to 11 because the global search space for size 18 and 20 is huge, the B&B solver needs to explore larger sub-trees in order to achieve good performances.

The B&B is deterministic, thus, it does not require multiple runs. Alternatively, when the execution time of COBBIGA is limited, it is a stochastic method because of the stochastic behavior of the islands of GAs. Indeed, the performances of the B&B islands depends on the efficiency of the GA islands in localizing good quality solutions in each interval. Thus,

GA Pop.Size	GA Stop.criterion	Mig.Freq to B&B islands
100	empty pop.received	60
Nb.Migrants	**Cross.Rate**	**Mut.Rate**
100	1	0.4

Table 6.1: Global settings used for the islands of GA in COBBIGA.

multiple independent runs are required for COBBIGA. The results shown in Table 6.2 are average results obtained with 6 independent runs of the parallel COBBIGA.

In Table 6.2, the number of islands (column 3) represents the number of B&B islands used in the parallel B&B model and the number of B&B and GA islands (each B&B island is collaborating with one GA) used in the parallel COBBIGA algorithm. The best found solution is given in column 6 and the real time (in seconds) taken to find it, in column 7. In order to give equal chances to the B&B solvers both in the pure B&B model and in COBBIGA, the same initial upper-bounds are used (column 5).

We notice that after the fixed execution time period (2 or 5 hours), the B&B algorithm alone was not able to provide a better solution while with the hybrid algorithm COBBIGA the best known solutions were found for all the considered benchmarks in less than one hour. It is important to emphasize that the best found solutions in COBBIGA are generally found by the B&B islands while exploring the promising regions pointed by the GA islands.

Nevertheless, we expected that the parallel COBBIGA model explores more intervals than the B&B alone because of the earlier improvement of the upper-bound in the B&B. This impact is not apparent as the percentage of unexplored intervals is huge for both models. This is probably due to the quality of the initial upper-bound and to the impact of the parallelism on the B&B algorithm. In order to check the impact of the initial upper-bound, we used ∞ as a starting solution for the last benchmark: Nug15. Nevertheless, after a $5h$ execution on 720 cores, the percentage of the unexplored search-space is the same for both algorithms. At this scale, it is difficult to measure the impact of each factor on the achieved speed up.

6.4.4.2 Proving the optimality of the solutions found using COBBIGA

After comparing the performances of the two parallel algorithms COBBIGA and B&B alone, we were interested in solving to optimality some benchmarks and hopefully, prove the optimality of some of the solutions found previously using COBBIGA. For this step, COBBIGA could also be used as an exact method if the stopping criterion is the end of the exploration in all the couples B&B-GA. However, since the objective of this step is only to prove the optimality of the best found solutions in the previous step of the experimentations, all the computing cores should be occupied by B&B solvers.

Thanks to the use of a memory based on the intervals in which only remaining sub-spaces

Bench.	Algo.	Islands	Cores	Init. sol.	Best	Avg.T(s)	Left%
Nug14b	COBBIGA	14	200	8140	**8084**	**782**	98
	B&B	7	200	8140	8140	8040	98
Had14	COBBIGA	14	256	37726	**37598**	**349**	96
	B&B	7	256	37726	37726	7315	96
Had15	COBBIGA	18	640	43396	**42860**	**725**	99
	B&B	9	640	43396	43396	19092	99
Nug16a	COBBIGA	16	640	15372	**15132**	**3951**	88
	B&B	8	640	15372	15372	18013	88
Nug15	COBBIGA	30	720	∞	**2230**	**1237**	99
	B&B	15	720	∞	2620	18000	99

Table 6.2: Results obtained using the parallel hierarchical master-slave model in both COBBIGA and B&B while solving the Q3AP benchmarks Nug14a, Had14, Had15, Nug15 and Nug16a.

are recorded in COBBIGA, the sub-spaces solved using COBBIGA in the previous stage are not re-explored in this step. Mainly, the B&B islands (without cooperation with GAs) are initialized with the best found solution and the remaining sub-intervals from the 2 or 5 hours execution of COBBIGA in the previous step of experiments.

The machines involved in these experiments belong to the sites Nancy and Sophia-Antipolis in Grid'5000. We used the clusters Griffon and Grellon from Nancy and the cluster Suno from Sophia-Antipolis. Each experiment involves only cores from clusters of the same Grid'5000 site in order to avoid the overhead due to inter-site communications in the grid. Because the implementation in ParadisEO is based on a version of MPI (MPICH-II) which is not fault tolerant, all the machines used in the experiments were reserved in advance, the besteffort mode can not be used.

In the case of Nug15, it was necessary to make several reservations on different number of cores with different time periods because of the variable availability of the grid resources (the grid is shared with other users). The intermediate results including the best found solution and the remaining intervals to explore, are updated regularly and used later to restart the algorithm. On average, 300 cores, from Grid'5000 site Sophia-Antipolis, were used to solve Nug15 with a pic of 720 cores.

For the rest of the benchmarks one reservation was enough. The results of these experiments are given in Table 6.3. The best found solutions for the benchmarks Had12, Had12b, Nug12c, Had14, Nug15 are all proved to be optimal. Notice that the optimal values of the benchmarks Nug12 and Nug13 are already known from the literature [47] and corresponds to the best found solutions using COBBIGA.

We notice that some benchmarks have the same size but do not require the same

Bench.	Initial sol.	Optimal	Time	Cores
Had12	19430	19430	2008s	440
Had12b	21128	21128	67s	440
Nug12	580	580	1780s	200
Nug12c	1326	1326	7429s	200
Had14	37598	37598	18h42mn	528
Nug15	2230	2230	9 days 7hours and 40 min.	Avrg. 300

Table 6.3: Solving some benchmarks to optimality

execution time. This is very common in B&B because the time required to solve each node depends on the bounding and pruning operations which are related to the structure of the benchmark. For example, the benchmarks Nug12 and Had12 of size 12 require approximately the same execution time, while Had12b is remarkably easy to solve and Nug12c is more difficult.

The resolution of Nug15 is repeated using the platform B&B@Grid (see Section 6.4.2.4) on thousands of cores in Grid'5000. The statistics of the experimentation are given in Table 6.4. This experiment is used in order to confirm the optimal value obtained by the parallel B&B islands in COBBIGA. The average number of cores involved in this experiment is around 1123 with a pic of 3427 belonging to 6 Grid'5000 sites. The experimentation lasted 11 days and the reservation mode used in Grid'5000 is besteffort (the platform B&B@Grid is fault-tolerant).

We notice that COBBIGA performed better execution time than B&B@Grid for Q3AP (9 days against 11 days), on less machines (average 300 cores against average 1123 cores). This is explained by the fact that the B&B@Grid adapted to Q3AP still needs some improvements in order to be more efficient for Q3AP. Indeed, the efficiency of a parallel B&B for Q3AP depends on several parameters. Among them, the number of lower-bound calculations. If the granularity of the nodes to be solved by the simultaneous B&B processes is too small, many expensive redundant bounding calculations will be done in each worker in B&B@Grid, because the lower-bound of a given Q3AP node requires the computation of the lower-bounds of all the nodes in its path. On the other hand, unlike the bounding function of the permutation flowshop, the Q3AP lower-bound is CPU time intensive. For this reason, in the implementation of the parallel COBBIGA we tried to explicitly control the size of the nodes to be solved by the B&B solvers. This is done by choosing a static decomposition of the global interval in such a way to avoid small size nodes (please refer to Section 6.3.2.2).

In the state-of-the-art of exact methods applied to Q3AP (Section 6.4.1), the largest benchmark size solved to optimality was limited to 13 [48] using a sequential B&B algorithm. In a recent work, Galea and Le Cun proposed a parallelization of the lower-bound function in the BOB++ B&B solver for Q3AP in [39]. The parallelization of the lower-bound coupled with a coarse grain parallelism of the search tree leads to a better execution time for Nug15 on shared memory machines. Indeed, Nug15 is solved in approximately 5 days instead of 9

days.

Our parallelization approach for large scale computational grids is based on a sequential version of the BOB++ B&B solver. Indeed, the exact solver is used as a black box in COBBIGA. Thus, the two parallelizations are complementary and better performances may be reached if the distributed parallelism over the grid is coupled to a local parallelism in each Grid'5000 multi-core machine.

Bench.	Nug15 (Q3AP)
Initial sol.	2233
Optimal sol.	2230
Time (s)	961917 (11d/3h/11m/57s)
Avrg. cores	1123
Max. cores	3427
Sites	Lille, Grenoble, Sophia, Bordeaux, Toulouse, Rennes et Orsay

Table 6.4: Experiment on Nug15 using B&B@Grid.

6.4.4.3 Performance evaluation of the parallel COBBIGA as a metaheuristic

The third and last step in the experimental process consists in comparing the parallel COBBIGA as a heuristic method to a hierarchical parallel hybrid GAILS. Because standard GAs without any hybridization with local search techniques have been proved inferior to GAs with hybridization in many analyses in the literature, the algorithm proposed for comparison in order to evaluate the performance of the parallel COBBIGA is a parallel hybrid GA-ILS which consists of a standard GA in which an ILS algorithm is incorporated to the mutation operator. Furthermore, as the initial population of GAs influences the quality of the final population, we also used the interval-based initialization method on each GAILS island. Thus, COBBIGA is compared to an enhanced hybrid pure metaheuristic.

30 runs are performed using the two algorithms (COBBIGA and GAILS) on the following benchmarks: Nug12, Nug13, Nug15, Nug16a, Nug18 and Nug20 using equal number of resources: 300 cores belonging to the Grid'5000 sites Sophia-Antipolis and Grenoble. The results are given in Table 6.5 where the column "Hits" refers to the percentage of runs in which each one of the algorithms finds the optimal or best known value.

Because there is no guaranty that GAILS will find the best solution in all the runs, we run the COBBIGA first until it reaches the best solution then we use the worst time required by COBBIGA as a time limit for GAILS. The average, best and worst times given for GAILS are calculated only on the percentage of runs in which it finds the best solution.

We notice that unlike the pure metaheuristic GAILS, COBBIGA always succeeds to find the optimal solution in less than two hours for all the benchmarks. Unlike the parallel

COBBIGA, the parallel GAILS fails in 9,63% of the runs for Nug13, 38% for Nug18, and in 8% of the runs in the case of the benchmark Nug20. Regarding the rest of the benchmarks (Nug12, Nug16a, Had14 and Nug15), the parallel COBBIGA is equivalent to the parallel GAILS as both algorithms succeed to find the best solution in 100% of the runs.

However, generally COBBIGA takes more time to localize the optimal solution compared to the parallel hybrid GAILS (when it finds the best solution) because of the use of a B&B solver in the former. In Table 6.5, the columns 6, 7 and 8 refer respectively, as their names indicate it, to the average time among 30 runs required by each algorithm to reach the optimum (or the best known solution), the best and the worst time in seconds. Note that the reported execution times are the real times on dedicated machines.

Globally, the results indicate that the parallel COBBIGA is more robust than the parallel GAILS algorithm but it is more CPU time consuming. Indeed, if the parallel GAILS do not get the best solution in the earlier generations, the algorithm will converge and the best solution in the population will not be improved even if it runs for a long time period, unless some diversification mechanisms are used. Alternatively, in COBBIGA, the best solution will be found sooner or later because (1) the population of the GA islands are renewed regularly in order to look unexplored sub-spaces, (2) the B&B islands intensify the search around good quality solutions and (3) the use of the intervals to parallelize the cooperative hybrid method enables to spread the global search over diverse sub-spaces that are explored in parallel without any redundancy.

Bench.	Algo.	Avg. Fitness	Hits(%)	Best Known	Avrg. Time	Best time (s)	Worst time (s)
Nug12	COBBIGA	580	100	580*	229	161	356
	Island GAILS	580	100		52	31	88
Nug13	COBBIGA	1912	100	1912*	1002	202	3742
	Island GAILS	1912	91		578	39	5042
Nug15	COBBIGA	2230	100	2230*	1495	378	3522
	Island GAILS	2230	100		2094	200	2094
Nug16a	COBBIGA	15132	100	15132	1528	416	4364
	Island GAILS	15132	100		2100	289	
Nug18	COBBIGA	17836	100	17836	4223	611	12945
	Island GAILS	17836	62		722	133	1974
Nug20	COBBIGA	25590	100	25590	2878	660	5876
	Island GAILS	25590	92		1543	307	2286

Table 6.5: Results obtained using parallel hierarchical COBBIGA and GAILS on the Q3AP benchmarks Nug12, Nug13, Nug15, Nug16a, Nug18 and Nug20.

6.5 Conclusion

In this chapter, we presented parallelization strategies for the hybrid algorithms COBBIGA and HGABB/HAGABB. These strategies are based on some space decomposition techniques used for parallel B&B algorithms. That is the distribution of sub-trees among a large number of computing resources. Because the interval representation is now implemented both in B&B and GAs, we re-used the interval-based decomposition proposed by Mezmaz in [76] to parallelize B&B algorithms for permutation problems, in the parallelization of the hybrid metaheuristics HGABB/HAGABB and COBBIGA.

Regarding the implementation, two different C++ frameworks were used in the implementation of the metaheuristics and the B&B algorithm, the frameworks ParadisEO and BOB++ respectively. The parallel hybrid algorithms COBBIGA and HGABB/HAGABB are implemented using a coupling algorithm that we called ParadisEO-HEM (Hybrid Exact Metaheuristics). This hybridization framework is implemented as an extension for ParadisEO in order to facilitate the design and implementation of hybrids combining ParadisEO-based metaheuristics with exact methods implemented using extern C++ frameworks. Low-level and high-level hybridizations are possible using ParadisEO-HEM for any kind of problems.

In order to evaluate the performances of the interval-based parallelization proposed for COBBIGA, several experiments were conducted on the international computational grid, Grid'5000. The parallel COBBIGA algorithm is deployed and run on hundreds of processors belonging to Grid'5000 in order to efficiently find near-optimal solutions for large Q3AP benchmarks and optimal ones for moderate size benchmarks. Indeed, the parallel COBBIGA is used to solve the size 15 Q3AP benchmark (Nug15) to optimality in two steps. First, COBBIGA is executed during 5 hours. Then, the GA islands in COBBIGA are stopped and all the processing resources were allocated to the B&B islands which are initialized with the best solution achieved in the first step and the remaining intervals, those which are not optimally solved in the first step. Thus, the execution of COBBIGA offers two advantages: the achievement of good quality upper-bounds for the B&B solver and the non redundancy of the search as the regions of the search space, which are explored in the first step, are not reconsidered in the second step. The parallel COBBIGA is also compared to an enhanced pure parallel hybrid metaheuristic combining a GA and the ILS algorithm. The parallel COBBIGA, even if used as a heuristic method in those experiments, achieved 100% of successful runs on all the experimented benchmarks while the parallel GAILS metaheuristics, even if it is also very efficient, does not succeed on all the benchmarks.

Part III

Conclusions

Chapter 7

Conclusions and future research

Contents

7.1 Summary

During this thesis we studied different challenging permutation-based problems, among them assignment problems such as QAP and Q3AP. Some permutation problems are among the hardest optimization problems. This difficulty is partially due to the combinatorial explosion which occurs when the size of the problems increases. This makes large instances very difficult to solve with standard optimization algorithms. For example, the size of the largest QAP benchmarks solved to optimality is around 30 [48] and the size of the largest Q3AP benchmark solved to optimality has just reached 16 [39], while real life applications for Q3AP in some telecommunication protocols are of size 16, 32 and 64.

Our goal was to design a hybridization software for easy implementation of hybridization strategies combining metaheuristics with tree-based exact search, in order to solve larger permutation-based problems more efficiently.

To this end, the work carried out during this thesis is summarized in the following points:

– We studied the properties of a variety of permutation-based problems from different fields: assignment problems (AP3, QAP and Q3AP), scheduling problems (permutation flowshop and JSP) and the TSP. Because each problem uses a different interpretation of the permutation representation, we classified the problems according to this criterion into three classes. In absolute position problems, the absolute position of elements in the permutation are the only matter. This class includes all assignment problems. In relative order problems, the permutation is used as a sequence, the relative order between elements is the only meaningful information. This class includes TSP and permutation flowshop problem. Finally, in the class absolute

order both the order and absolute positions of elements are important. This class includes the JSP. This classification is completed with a review of state-of-the-art neighborhood and genetic operators used for each class of permutation-based problems.

– For the design of low-level and high-level hybridization strategies between metaheuristics and tree-based exact methods, our strategy consisted of using the same representation of the search space in both sides for a more accurate cooperation between the different methods. The idea was to generalize the B&B tree to metaheuristics. This tree is used in B&B in order to enumerate the solution space of the tackled problem. Moreover, in order to efficiently parallelize this algorithm over computational grids, an encoding is proposed in [76] that represents each set of contiguous nodes by an interval. The use of this encoding in metaheuristics will help to find some connections between the divergent search strategies used in metaheuristics and B&B.

– The generalization of the interval encoding to metaheuristics requires the use of some basic mathematical properties of permutations such as inversions, inversion tables, Lehmer codes and some algorithms from the discrete mathematics community for the mapping of permutations to the discrete space of natural numbers. After studying the existing algorithms in the literature, we proposed new mapping algorithms for generalized permutation problems (problems using more than one permutation to represent solutions). Those algorithms could also be exploited in other domains for other purposes, as they are not related to metaheuristics.

– Once the tree representation and the concept of interval are adapted to metaheuristics, interval-based neighborhood and genetic operators for permutation-based problems were designed. Those operators are used to search restricted sub-spaces delimited by a given interval. We also analyzed the re-usability of such operators for each class of permutation-based problems. The interval-based operators are used to design efficient low-level and high-level hybridization strategies combining metaheuristics with B&B.

– Indeed, two hybridization schemes based on the B&B tree are proposed: the Hybrid GA-B&B (HGABB) and its adaptive version Hybrid Adaptive GA-B&B, and a cooperative asynchronous algorithm COBBIGA . In HGABB/HAGABB, The B&B is embedded into a mutation operator in order to optimally search large sub-spaces. Some mechanisms from other metaheuristics are used to escape optimally solved sub-spaces when the GA reaches convergence: a sort of tabu list of already explored sub-spaces represented as intervals is maintained inside the mutation operator. This allows to escape optimally solved sub-spaces during the previous generations. COBBIGA is a high-level asynchronous cooperative approach based on interval-based space decomposition techniques used both in B&B and in GA.

– Because such complex algorithms are CPU time intensive, we proposed a parallelization of the two algorithms COBBIGA and HGABB/HAGABB by using some space decomposition techniques inspired from parallelization strategies of the B&B algorithm and other tree-based search algorithms.

– From the implementation point of view, for easy implementation of other hybridization strategies between metaheuristics and exact search algorithms developed using different frameworks, we developed ParadisEO-HEM, a C++ hybridization framework integrated to ParadisEO. In order to evaluate the performances of the proposed methods, a case study on the Q3AP is done and new near-optimal solutions for large unsolved benchmarks are found with a high reliability.

7.2 Future research

Some future research directions have been reported earlier in this dissertation.

The first future research suggestion concerns the use of a transposition order in the permutation tree instead of a lexicographic one in order to obtain more efficient interval-based operators. Indeed, if the transposition order is used, every set of solutions covered by an interval, no matters its size, forms a highly connected 2-exchange neighborhood. This allows every solution in the interval to be reached by the restricted search operators. Nevertheless, the use of a transposition order requires the definition of new mapping algorithms from the permutation space to the space of natural numbers. Furthermore, the development of such mapping functions is a big challenge for the discrete mathematics community as such algorithms could have other applications in other fields.

Concerning the hybridization strategies between GAs and B&B, it would be interesting to make a theoretical study on the cooperative COBBIGA in order to give a bound for the execution time or the percentage of the space that has to be explored to guarantee that the final solution is optimal. Or at least, give an estimation on the relative error expected. Indeed, COBBIGA is in the same time a metaheuristic and an exact algorithm, depending on the stopping criterion used. Thus, the optimal solution will be found sooner or later. However, from the metaheuristic point of view, there is no guarantee because the algorithm contains a stochastic part (a GA). Nevertheless, the promising results obtained on standard benchmarks of Q3AP encourage us to complete the model by some theoretical estimations of the performances of the algorithm and the study of optimal values for the different parameters involved.

In order to complete the evaluation of the hybridization framework and the different strategies developed in this thesis, we also consider the application of the same algorithms to other permutation-based problems.

Finally, regarding the space decomposition parallelization strategies used in the hybrid schemes, the adequate number of GA islands searching disjoint intervals should be determined. Indeed, unlike exact methods, in metaheuristics the adage "divide and conquer" is not really applicable or at least there is a limit beyond which the performances of the search will decrease. In divide and conquer exact algorithms, the search is based on enumeration. Thus, the more we share the search space, the better will be the speed up. Of course, as long as we respect some constraints related to the underlying parallel architecture (latency of communications) and others related to the problem at hand (the cost of bounding functions). In

metaheuristics, the search strategy consists of constructing search paths through iteratively constructed neighborhoods, in such a way to reach near-optimal solution by examining only a selected number of solutions in the entire search space. The decomposition of the solution space on several parallel searches would improve the global search efficiency because of the diversity of the explored sub-spaces. However, using fine grain decomposition could lead to a poor metaheuristic if no cooperation mechanisms are used. In between of the two extremes, one may find the adequate grain size. This is our short-term future work, it can be done by extensive experiments.

Publications

[CEC2010] Malika Mehdi, Nouredine Melab, El-Ghazali Talbi and Pascal Bouvry, Interval-based Initilialization Method for Permutation-based Problems, Proceedings of the World Congress on Computational Inteligence. Barcelona, Spain, 2010.

[GECCO2011] Malika Mehdi, Jean-Claude Charr, Nouredine Melab, El-Ghazali Talbi, and Pascal Bouvry, A cooperative tree-based hybrid GA-B&B approach for solving challenging permutation-based problems, Proceedings of the 13th annual conference on Genetic and evolutionary computation. Pages 513-520. Dublin Ireland, 2011. **BEST PAPER AWARD**.

[JGUC2009] Malika Mehdi, Mohand-Said Mezmaz, Nouredine Melab, El-Ghazali Talbi and Pascal Bouvry, P2P computing for large tree exploration-based exact optimisation, International Journal of Grid and Utility Computing, Volume 1, Issue 3, pages 252-260. August 2009.

[IJFCS2011] Lakhdar Loukil, Malika Mehdi, Nouredine Melab, El Ghazali Talbi and Pascal Bouvry, Parallel Hybrid Genetic Algorithms for Solving Q3AP on Computational Grid, International Journal of Foundations of Computer Science, World Scientific Publisher, In Press, 2011.

References

[1] K. Aida and W. Natsume. Distributed computing with hierarchical master-worker paradigm for parallel branch and bound algorithm. *Proc. of IEEE International Symposium on Cluster Computing and the Grid*, pages 156–163, 2003. 30

[2] R. M. Aiex, M. G. C. Resende, P.M. Pardalos, and G. Toraldo. Grasp with path relinking for the three-index assignment problem. 2000. 52

[3] Enrique Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7 – 13, 2002. URL http://www.sciencedirect.com/science/article/pii/S0020019001002812. 32

[4] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002. 19, 23, 32, 33

[5] Enrique Alba and José M. Troya. Influence of the migration policy in parallel distributed gas with structured and panmictic populations. *Applied Intelligence*, 12(3):163–181, Mai 2000. 33

[6] Enrique Alba and José M. Troya. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. *Future Generation Computer Systems*, 17(4):451–465, January 2001. 34

[7] Eric Angel and Vassilis Zissimopoulos. On the hardness of the quadratic assignment problem with metaheuristics. *Journal of Heuristics*, 8:399–414, July 2002. ISSN 1381-1231. doi: 10.1023/A:1015454612213. URL http://portal.acm.org/citation.cfm?id=594956.595094. 44

[8] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Software: Practice and Experience*, 32(15):1437–1466, 2002. ISSN 1097-024X. URL http://dx.doi.org/10.1002/spe.488. 139

[9] M. Ben-Daya and M. Al-Fawzan. A tabu search approach for the flow shop scheduling problem. *European Journal of Operational Research*, 109(1):88 – 95, 1998. URL http://www.sciencedirect.com/science/article/pii/S0377221797001367. 52

[10] Christian Bierwirth, Dirk Mattfeld, and Herbert Kopfer. On permutation representations for scheduling problems. 1141:310–318, 1996. URL http://dx.doi.org/10.1007/3-540-61723-X_995. 3, 49, 52, 54, 128

[11] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, September 2003. 9

[12] Vincent Boyer, Didier El Baz, and Moussa Elkihel. Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound. *European J. of Industrial Engineering*, 4(4):434–449, 2010. 27

[13] Peter Brucker. *Scheduling Algorithms*. Springer, 2007. 48, 49

[14] R. E. Burkard, S. E. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *European Journal of Operational Research*, 55:115–119, 1991. QAPLIB is found on the web at http://www.seas.upenn.edu/qaplib. 6, 90, 149, 157

[15] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, 2009. 43

[16] R.E. Burkard and R. Rudolf. Computational investigations on 3-dimensional axial assignment problems. *Belgian Journal of Operations Research, Statistics and Computer Science*, 32:85–98, 1992. 45

[17] S. Cahon, N. Melab, and E.-G Talbi. ParadisEO: a Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10:353–376, 2004. 5, 118, 150

[18] Erick Cant-paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles*, 10, 1998. 32, 33

[19] Erick Cantú-Paz and David E. Goldberg. Modeling idealized bounding cases of parallel genetic algorithms. In *In*, pages 353–361. Morgan Kaufmann Publishers, 1996. 34

[20] Erick Cantu-Paz and David E. Goldberg. Efficient parallel genetic algorithms: Theory and practice. *Computer Methods in Applied Mechanisms and Engineering*, 186:221–238, 2000. 19, 33

[21] Jean-Louis Bouquard Cedric Pessan and Emmanuel Neron. Genetic branch-and-bound or exact genetic algorithm? In *Lecture Notes In Computer Science*, volume 4926, pages 136–147, 2008. 2, 27, 28, 104

[22] Phillipe D'Anfray and Emmanuel Jeannot. GRID'5000 une plateforme d'expérimentation pour les systèmes distribués à large échelle. In *Journées Réseaux - JRES 2007*, Strasbourg, France, 2007. URL http://hal.inria.fr/inria-00181446/en/. 156

[23] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991. 56

[24] D. Debels and Mario Vanhoucke. Meta-heuristic resource constrained project scheduling: solution space restrictions and neighbourhood extensions. Vlerick leuven gent management school working paper series, Vlerick Leuven Gent Management School, 2006. URL http://econpapers.repec.org/RePEc:vlg:vlgwps:2006-18. 141

[25] A. di Costanzo. *Branch and bound with peer-to-peer for large scale grids*. PhD thesis, Université de Sophia Antipolis, 2007. 30

[26] Hopkins-J.W. Dickey, J.W. Campus building arrangement using topaz. *Transportation Research*, 6:59–68, 1972. 43

[27] A. Djerrah, Bertrand Le Cun, Van-Dat Cung, and Catherine Roucairol. Bob++: Framework for solving optimization problems with branch-and-bound methods. In *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing, HPDC-15, Paris, France*, 2006. 30, 118, 150, 151

[28] Zvi Drezner. Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Comput. Oper. Res.*, 35:717–736, March 2008. ISSN 0305-0548. doi: 10.1016/j.cor.2006.05.004. URL http://portal.acm.org/citation.cfm?id=1287844.1288003. 19, 23

[29] Mohammed El-Abd and Mohamed Kamel. A taxonomy of cooperative search algorithms. 3636:902–902, 2005. URL http://dx.doi.org/10.1007/11546245_4. 25, 140

[30] Alwalid N. Elshafei. Hospital layout as a quadratic assignment problem. *Operational Research Quarterly*, 28(1):167–179, 1977. URL http://www.jstor.org/stable/3008789. 43

[31] Charles Fleurent, Jacques, and Jacques A. Ferland. Genetic hybrids for the quadratic assignment problem. In *DIMACS Series in Mathematics and Theoretical Computer Science*, pages 173–187. American Mathematical Society, 1993. 23

[32] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11:115–128, 1996. 139

[33] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15:200–222, August 2001. ISSN 1094-3420. 139

[34] Alan P. French, Andrew C. Robinson, and John M. Wilson. Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7(6):551–564, 2001. 2, 26, 28, 104

[35] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002. 139

[36] A. M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161 – 164, 1983. URL http://www.sciencedirect.com/science/article/pii/0377221783900784. 46, 48

[37] A. M. Frieze and J. Yadegar. An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *The Journal of the Operational Research Society*, 32(11):989–995, 1981. URL http://www.jstor.org/stable/2581852. 45

[38] Crainic T. G. and Toulouse M. *Parallel Strategies for Metaheuristics*. 2002. 13, 14, 25, 30, 31, 32, 140

[39] F Galea and B LeCun. A parallel exact solver for the three-index quadratic assignment problem. In *In proceeding of IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Mai 2011. 6, 149, 160, 166

[40] F. Galea, P.M Hahn, and B. LeCun. A parallel implementation of the quadratic three-dimensional assignment problem using the Bob++ framework. *The 21st Conference of the European Chapter on Combinatorial Optimization (ECCO XXI)*, 2008. 6, 90, 118, 149, 150, 151, 154

[41] José E. Gallardo, Carlos Cotta, and Antonio J. Fernández. On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 37:77–83, 2007. 2, 27, 28, 104, 105

[42] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. ISBN 0716710455. 1, 10, 44, 46, 49

[43] Bernard Gendron and Teodor Gabriel Crainic. Parallel branch-and-bound algorithms: Survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994. URL http://www.jstor.org/stable/171985. 29

[44] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989. 2, 19, 20, 104

[45] D. Goldberg and R. Lingle. Alleles, loci, and the traveling salesman problem. In *Proceedings of the First International Conference on Genetic Algorithms and their Applications (ICGA'85*, pages 154–159, Cambridge, 1985. 19, 55, 78

[46] K.Miettinen H. Maaranen and M.M. Makela. Quasi-random initial population for genetic algorithms. *Computers and Mathematics with Applications*, 47:1885–1895, 2004. 20

[47] P. M. Hahn, B.-J. Kim, T. Stutzle, S. Kanthak, W. L. Hightower, Z. Ding H. Samra, and M. Guignard. The quadratic three-dimensional assignment problem: Exact and approximate solution methods. *European Journal of Operational Research*, 184:416–428, 2008. 46, 47, 52, 53, 118, 149, 154, 159

[48] Peter M. Hahn, Yi-Rong Zhu, Monique Guignard, and J. MacGregor Smith. Exact solution of emerging quadratic assignment problems. *International Transactions in Operational Research*, 17(5):525–552, 2010. ISSN 1475-3995. doi: 10.1111/j.1475-3995.2010.00763.x. URL http://dx.doi.org/10.1111/j.1475-3995.2010.00763.x. 6, 149, 160, 166

[49] Olivier C. Martin Helena R. Lourenço and Thomas Stutzle. Iterated local search. *International Series in Operations Research and Management Science*, 57:320–353, 2003. 18, 88, 107

[50] John H. Holland. Outline for a logical theory of adaptive systems. *J. ACM*, 9:297–314, July 1962. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321127.321128. URL http://doi.acm.org/10.1145/321127.321128. 18

[51] John H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-58111-6. 18

[52] Seog-Moon Oh Ji-Won Chung and In-Chan Choi. A hybrid genetic algorithm for train sequencing in the korean railway. *Omega The International Journal of Management Science*, 37:555–565, 2009. 20

[53] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954. URL http://dx.doi.org/10.1002/nav.3800010110. 48

[54] Selmer M. Johnson. Generation of permutations by adjacent transposition. *Mathematics of Computation*, 17(83):282–285, 1963. ISSN 00255718. URL http://www.jstor.org/stable/2003846. 41, 84

[55] L. Jourdan, M. Basseur, and E.-G. Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620 – 629, 2009. URL http://www.sciencedirect.com/science/article/pii/S0377221708003597. 25

[56] Hillol Kargupta, Ka. Lyanmoy Deb, and David E. Goldberg. Ordering genetic algorithms and deception. In *Parallel Problem Solving from Nature PPSN II*, pages 47–56. Springer, 1992. 3, 19, 50, 54, 55, 56

[57] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998. ISBN 1558604758. 139

[58] B.-J. Kim. *Investigation of methods for solving new classes of quadratic assignment problems (QAPs)*. PhD thesis, University of Pennsylvania, 2006. 12, 149

[59] Donald Knuth. *The Art of Computer Programming, Volume 2*. 1997. 37, 39

[60] Zdenek Konfrst. Parallel genetic algorithms: Advances, computing trends, applications and perspectives. *Parallel and Distributed Processing Symposium, International*, 7: 162b, 2004. doi: http://doi.ieeecomputersociety.org/10.1109/IPDPS.2004.1303155. 19, 32

[61] T. C. Koopmans and M. J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957. 43

[62] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002. URL http://dx.doi.org/10.1002/spe.432. 139

[63] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. ISSN 1931-9193. doi: 10.1002/nav.3800020109. URL http://dx.doi.org/10.1002/nav.3800020109. 42

[64] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13:129–170, 1999. 54

[65] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966. 2, 10, 104

[66] Eugene L. Lawler. The quadratic assignment problem. *Management Science*, 9(4): 586–599, 1963. URL http://www.jstor.org/stable/2627364. 44

[67] Yong Li, Panos M. Pardalos, and Mauricio G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. 1994. 16

[68] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973. 52

[69] Piotr Lipinski. A hybrid evolutionary algorithm to quadratic three-dimensional assignment problem with local search for many-core graphics processors. 6283:344–351, 2010. 150

[70] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657 – 690, 2007. ISSN 0377-2217. doi: DOI:10.1016/j.ejor.2005.09.032. URL http://www.sciencedirect.com/science/article/pii/S0377221705008337. 43

[71] Lakhdar Loukil, Malika Mehdi, Nouredine Melab, El-Ghazali Talbi, and Pascal Bouvry. A parallel hybrid genetic algorithm-simulated annealing for solving q3ap on computational grid. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, 2009. 150

[72] D. Magos and P. Miliotis. An algorithm for the planar three-index assignment problem. *European Journal of Operational Research*, 77(1):141 – 153, 1994. URL http://www.sciencedirect.com/science/article/pii/0377221794900345. 45

[73] Roberto Mantaci and Fanja Rakotondrajao. A permutations representation that knows what eulerian means. *Discrete Mathematics & Theoretical Computer Science*, 4(2):101–108, 2001. 38

[74] T. Mautor and C. Roucairol. Difficulties of exact methods for solving the quadratic assignment problem. *Series in Discrete Math. Theoret. Comput. Sci.*, 16, 1994. 44

[75] N. Melab. Contributions à la résolution de problèmes d'optimisation combinatoire sur grilles de calcul. HDR, Novembre 2005. 29

[76] M. Mezmaz, N. Melab, and E-G. Talbi. A grid-enabled branch and bound algorithm for solving challenging combinatorial optimization problems. In *In Proc. of 21th IEEE Intl. Parallel and Distributed Processing Symp.*, Long Beach, California, March 2007. 2, 3, 4, 5, 12, 30, 61, 62, 70, 73, 114, 116, 142, 147, 154, 164, 167

[77] Mohand Mezmaz. *Methodes d'optimisation combinatoire sur grilles de Calcul*. PhD thesis, Lille university, 2007. 6, 153, 155

[78] Miron Livny Michael Litzkow and Matt Mutka. Condor: A hunter of idle workstations. In *IEEE 8th Intl. Conf. on Dist. Comp. Sys.*, 1988. 139

[79] I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 224–230, 1987. URL http://portal.acm.org/citation.cfm?id=42512.42542. 54

[80] Ibrahim Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, 63:511–623, 1996. URL http://dx.doi.org/10.1007/BF02125421. 13

[81] Eduardo L. Pasiliao. Local neighborhoods for the multidimensional assignment problem. In *Dynamics of Information Systems*, volume 40 of *Springer Optimization and Its Applications*, pages 353–371. Springer New York, 2010. ISBN 978-1-4419-5689-7. URL http://dx.doi.org/10.1007/978-1-4419-5689-7_19. 52

[82] Stefan Pesko. Differential evolution for small tsps with constraints. In *4th International Scientific Conference, Challenges in Transport and Communication*, September 2006. 5, 38, 63

[83] William Pierskalla. The tri-substitution method for the three dimensional assignment problem. *Canadian operational research society journal*. 45

[84] Gershoni N. Radday Y.T. Pollatschek, M.A. Optimization of the typewriter keyboard by simulation. *Angewandte Informatik*, 17:438–439, 1976. 43

[85] Antonin Ponsich and Carlos Coello Coello. Testing the permutation space based geometric differential evolution on the job-shop scheduling problem. 6239:250–259, 2011. URL http://dx.doi.org/10.1007/978-3-642-15871-1_26. 49

[86] Jakob Puchinger and Günther R. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Lecture Notes In Computer Science*, volume 3562, pages 41–53, 2005. 25, 26, 27

[87] Jakob Puchinger, Günther R. Raidl, and Martin Gruber. Cooperating memetic and branch-and-cut algorithms for solving the multidimensional knapsack problem. In *In Pro. The 6th Metaheuristics International Conference*, 2005. 2, 22, 27, 104, 105

[88] C. S. Rabak and J. S. Sichman. Using a-teams to optimize automatic insertion of electronic components. *Advanced Engineering Informatics*, 17(2):95–106, 2003. 43

[89] Gunther Raidl. A unified view on hybrid metaheuristics. 4030:1–12, 2006. 22, 24, 25, 105, 109

[90] S. Ronald. More distance functions for order-based encodings. In *In Proceedings of the IEEE Conference on Evolutionary Computation*, pages 558–563, 1998. 42

[91] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms (second edition)*. Springer-Verlag Berlin Heidlberg, 2006. ISBN 3-540-25059-X. 14, 15, 60, 64

[92] Ruben Ruiz and Thomas Stutzle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033 – 2049, 2007. 52

[93] Sartaj Sahni and Teofilo Gonzalez. P-complete approximation problems. *J. ACM*, 23: 555–565, July 1976. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/321958.321975. URL http://doi.acm.org/10.1145/321958.321975. 43, 44, 48

[94] Hamed Samarghandi, Pouria Taabayan, and Farzad Firouzi Jahantigh. A particle swarm optimization for the single row facility layout problem. *Comput. Ind. Eng.*, 58: 529–534, May 2010. ISSN 0360-8352. doi: http://dx.doi.org/10.1016/j.cie.2009.11.015. URL http://dx.doi.org/10.1016/j.cie.2009.11.015. 5, 63

[95] H. Samra, Z. Ding, and P. M. Hahn. Optimal symbol mapping diversity for multiple packet transmissions. In *Proceedings of 2003 International Conference on Acoustics, Speech and Signal Processing*, Hong Kong, 2003. 46

[96] Victor Robles Santiago Muelas, Jos M. Pena and Antonio LaTorre. Voronoi-initialization island models for solving real-coded deceptive problems. In *Genetic and Evolutionary Computation Conference Proceedings*, pages 993–1000, 2008. 20

[97] Hamid R. Tizboosh Shahryar Rahnamayan and Magdy M.A Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers and Mathematics with Applications*, 53:1605–1614, 2007. 20

[98] Zbigniew Skolicki and Kenneth De Jong. The influence of migration sizes and intervals on island models. In *Genetic and Evolutionary Computation Conference Proceedings*, pages 1295–1302. SIGEVO: ACM Special Interest Group on Genetic and Evolutionary Computation, June 2005. 33, 34

[99] Thomas Stutzle. *Local search algorithms for combinatorial problems: Analysis, algorithms and new applications*. PhD thesis, Technische Universitat Darmstadt, Germany. 18

[100] Thomas Stutzle. Applying iterated local search to the permutation flow shop problem. Technical report, Darmstadt, University of Technology, 1998. 52

[101] Thomas Stutzle. Iterated local search for the quadratic assignment problem. *European Journal of Operational Research*, 174(3):1519 – 1539, 2006. URL http://www.sciencedirect.com/science/article/pii/S0377221705002651. 52, 53

[102] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8:541–564, 2002. 11, 14, 16, 22, 23, 24, 25, 105, 109, 140

[103] H. F. Trotter. Algorithm 115: Perm. *Commun. ACM*, 5:434–435, August 1962. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/368637.368660. URL http://doi.acm.org/10.1145/368637.368660. 41, 84

[104] Luong Thé Van, Lakhdar Loukil, Melab Nouredine, and Talbi El-Ghazali. A GPU-based Iterated Tabu Search for Solving the Quadratic 3-dimensional Assignment Problem. In *Workshop on Parallel Optimization in Emerging Computing Environments (POECE) in Conjunction with the International Conference on Computer Systems and Applications (AICCSA)*, 2010. URL http://hal.inria.fr/inria-00520468/PDF/POECE2010Paper.pdf. 52, 150

[105] F. van den Bergh and A. P. Engelbrecht. A Cooperative approach to particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):225–239, 2004. URL http://dx.doi.org/10.1109/TEVC.2004.826069. 141

[106] Hamming R. W. Error detecting and error correcting codes. *Bell System Technical Journal*, 29:147–160, 1950. 42

[107] Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1999. 33

[108] L. Darrell Whitley. Cellular genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, 1993. 19, 34

[109] Butong Zhang, Xin Zhang, Jiancun Hu, and Dacheng Yang;. A hybrid arq scheme with constellation rearrangement and power adjustment. In *Proceedings of the International Conference of Wireless Communications, Networking and Mobile Computing*, pages 464–468, 2005. ISBN 0-7803-9335-X. 46